

Transient Reward Approximation for Grids, Crowds, and Viruses

Ernst Moritz Hahn, Holger Hermanns, Ralf Wimmer, and Bernd Becker, *Fellow, IEEE*

Abstract—We are interested in the analysis of very large continuous-time Markov chains (CTMCs) with many distinct rates. Such models arise naturally in the context security and dependability analysis of power grids, of computer virus vulnerability, and in the study of crowd dynamics. We use abstraction techniques together with novel algorithms for the computation of bounds on the expected final and accumulated rewards in continuous-time Markov decision processes (CTMDPs). These ingredients are combined in a partly symbolic and partly explicit (syblicit) analysis approach. In particular, we circumvent the use of multi-terminal decision diagrams, because the latter do not work well if facing a large number of different rates. We demonstrate the practical applicability and efficiency of the approach on two case studies.

Keywords—Continuous-time Markov chains, continuous-time Markov decision processes, abstraction, symbolic methods, OBDDs



1 INTRODUCTION

THE analysis of large Markov chains is a recurring challenge in many important areas ranging from quantitative security [1] to computer network dependability and performance [2], [3]. To evaluate properties of such systems, a standard approach is to perform numerical analysis, nowadays often embedded in a stochastic model checker [4], [5], [6], [7]. At its core, the model checker has to operate with a very large matrix induced by the Markov chain. In this context, the use of symbolic representations, in particular variations of decision diagrams, such as MTBDDs (multi-terminal decision diagrams) [8], [9], MDDs [10], or ZDDs [11] have made it possible to store and manipulate very large matrices in a symbolic manner. Many of the applications occurring in practice lead to very large continuous-time Markov chains (CTMCs) that nevertheless contain only a very small number of different transition rates. This is a primary reason why decision diagrams—where distinct rates are stored as distinct values in the structure—are effective. Whenever there are many pairwise different rates occurring, the decision diagram degenerates to a decision tree, and thus its size explodes. Therefore, models with a high number of different rates are a notorious problem for symbolic representations, and hence for the stochastic model checkers available to date.

However, there is a growing spectrum of important appli-

cations that give rise to excessive numbers of distinct rates. Power grid stability [12], [13], crowd dynamics [14], [15] as well as (computer) virus epidemiology [16], [17], [18] are important examples where Markov models are huge and rates change from state to state. The study of these phenomena is of growing importance for the assurance of security and dependability. Several of these examples can in some way be regarded as population models [19], [20], where the rates change with population counts, similar to models appearing in systems biology [21] and also in classical performance and dependability engineering [22], [23].

This paper targets the analysis of transient properties of CTMCs with both a large number of states as well as a large number of distinct transition rates. It presents a combination of abstraction techniques, an explicit representation of a small abstract model, and symbolic techniques—the latter using ordered binary DDs (OBDDs), not MTBDDs or MDDs—, which we thus call syblicit. The abstraction method relies on visiting all concrete states of the abstract model to obtain bounds on the transition matrix, but without having to store the state space explicitly. We also present ideas how to speed up this admittedly time-consuming process. On the one hand side, the approach can be seen as a continuation of our previous work on syblicit algorithms [24], [25]. On the other hand side, we harvest work done on abstraction of Markov Chains to abstract Markov chains or Markov decision processes [26], [27], [28], [29], [30], [31], [32].

A number of related methods exist: We build on many ideas for analyses using abstract Markov chains by Klink et al. [26], [28]. This paper extends their works by describing a widely applicable abstraction method, and also by handling more general transient properties.

MTBDD-based methods [8], [9] work well for some models, but have the disadvantages described above. The method of Wan et al. [10] uses a slightly different data structure to represent concrete models, and focuses on steady-state properties rather than transient ones. It does not rely on Markov decision models, and, though it works well in practice for certain model classes, cannot guarantee safe bounds for properties of the concrete

• Ernst Moritz Hahn is with the University of Oxford, United Kingdom.
E-Mail: emhahn@cs.ox.ac.uk. Part of this work was done while Ernst Moritz Hahn was with Saarland University.

• Holger Hermanns is with the Saarland University, Germany.
E-Mail: hermanns@cs.uni-saarland.de

• Ralf Wimmer and Bernd Becker are with the Albert-Ludwigs-University Freiburg, Germany.
E-Mail: {wimmer, becker}@informatik.uni-freiburg.de

This work was partly supported by the German Research Council (DFG) as part of the Transregional Collaborative Research Centre “Automatic Verification and Analysis of Complex Systems” (SFB/TR 14 AVACS) (see www.avacs.org for more information), the NWO-DFG bilateral project ROCKS, by grant agreement number 318490 SENSATION, the ERC Advanced Grant VERIWARE, and has received funding from the European Union Seventh Framework Programme under grant agreement number 295261 as part of the MEALS project.

model.

Techniques in which a symbolic representation of the transition matrix but an explicit values for each state of the concrete model needs to be stored, as for instance in the so called hybrid method [8], [33], or using other variants of DDs [11], and also methods using Kronecker representations [34], [35], [36], [37], [38] are more precise and might be faster than the method we propose. They are however not applicable in case the state space is excessively large, too large to store one value per state.

Smith [27] developed means for the compositional abstraction of CTMCs given in a process calculus. This way, he obtains an abstract Markov chain, which is then analysed by a method of Baier et al. [39] to obtain bounds for the time-bounded reachability probability. Our approach uses a different abstraction method and can handle a more general class of properties.

A paper by Buchholz [29] describes how bounds on long-run average (thus, non-transient) properties can be obtained from abstract Markov chains. It is based on a combination of policy and value iteration [40], and discusses the applicability of several variants of these methods on typical examples from queueing theory and performance evaluation.

The magnifying-lens abstraction [30] by de Alfaro et al. is similar to our approach in that it also builds on (repeated) visits of concrete model states without storing the whole concrete state space. It discusses a different model, discrete-time Markov decision processes (DTMDPs), and a different property, time-unbounded reachability probabilities.

D’Argenio et al. [41] discuss how DTMDPs given as MTBDDs can be abstracted to obtain a smaller abstract model, which is also a DTMDP, but small enough to be represented explicitly. In addition, a heuristic abstraction refinement method is presented. The target there was to obtain bounds for unbounded reachability probabilities. Works by Hermanns et al. [32] and by Kattenbelt et al. [31] later developed methods to use probabilistic games to provide tighter value bounds and predicate abstraction to handle larger or even infinitely large models, as well as refinement methods based on these frameworks. In contrast to the state-of-the-art for discrete-time models, the discussed refinement method we consider is more preliminary.

Other methods work with a finite subset of concrete states of the model under consideration, rather than subsuming concrete states in abstract ones. Doing so is possible for transient analyses, as there often the probability mass is concentrated on a small subset of states at each point of time, rather than being equally distributed among all states of the model. There exists a wide range of methods exploiting this fact in the analysis of CTMCs [42], [43], [44], [45]. Recently, this approach was extended to infinite-state Markov decision processes [46]. Here, two finite submodels are constructed which guarantee to bound the values over all policies from below and above. They can also be used to obtain a policy which is ε -optimal in the original model.

Such methods work well in case there is a moderate number of states with relevant probability, out of the very large number of all states the complete model consists of. If this is not

the case, these methods either need too much memory for storing the state information, or the result becomes imprecise because too much of the probability mass is not taken into consideration.

This paper is structured as follows: In Section 2, we provide basic notations and describe the symbolic data structures used for the later abstraction. We also describe the formal models we use, as well as the properties we are interested in. Section 3 describes algorithms to efficiently obtain an abstract model from a description of a concrete model, and discusses how they can be used to bound properties of the concrete model. In Section 4 we apply this method on two case studies, thus to show its practical applicability. Finally, Section 5 concludes the paper.

2 PRELIMINARIES

This section gives basic notations and formally defines the models and data structures that are used in the later parts of the paper.

A *distribution* over a finite or countable set A is a function $\mu: A \rightarrow [0, 1]$ such that $\sum_{a \in A} \mu(a) = 1$. By $\text{Distr}(A)$ we denote the set of all distributions over A .

The simplest stochastic model we consider is as follows:

Definition 1: A *discrete-time Markov chain (DTMC)* is a tuple $\mathcal{D} = (S, \mathbf{P})$ where

- S is a finite set of *states*,
- $\mathbf{P}: (S \times S) \rightarrow [0, 1]$ is the *probability matrix* such that $\sum_{s' \in S} \mathbf{P}(s, s') = 1$ for all $s \in S$.

By $X^{\mathcal{D}, s_0}: (\Omega_{\mathcal{D}} \times \mathbb{N}) \rightarrow S$ with $s_0 \in S$ we denote the unique stochastic process [47] of \mathcal{D} with initial state s_0 , where $\Omega_{\mathcal{D}}$ is the sample space to be used.

The time in a DTMC proceeds in discrete steps, and in each step a state change takes place. At step 0 the model starts in a given initial state $s_0 \in S$. The model moves to the next state, and will be in s_1 with probability $\mathbf{P}(s_0, s_1)$ for all $s_1 \in S$. From there, again the next state is chosen according to \mathbf{P} , and so on.

By Pr , we denote the *probability measure* on the measurable spaces $(\Omega_{\mathcal{D}}, \Sigma_{\mathcal{D}})$ of the DTMC \mathcal{D} under consideration, which is defined by the cylinder set construction over finite paths [48]. For instance, $\text{Pr}(X_n^{\mathcal{D}, s_0} = s_1 \vee X_{n+1}^{\mathcal{D}, s_0} = s_2)$ describes the probability that, having started in state s_0 , in step n we are in s_1 or in step $n+1$ we are in s_2 . For a measurable function $X: \Omega_{\mathcal{D}} \rightarrow \mathbb{R}$ we thus also have an *expectation* $\mathbf{E}(X) \stackrel{\text{def}}{=} \int_{\Omega_{\mathcal{D}}} X(\omega) \text{Pr}(d\omega)$. For instance, consider $X \stackrel{\text{def}}{=} \sum_{i=0}^{n-1} (f \circ X_i^{\mathcal{D}, s_0})$ such that $f(s_1) \stackrel{\text{def}}{=} 1$ and $f(s) \stackrel{\text{def}}{=} 0$ else. Then $\mathbf{E}(X)$ denotes the average number of steps within the first n steps in which the DTMC is in s_1 , under the condition that we started in s_0 .

We now discuss our basic stochastic model described informally in the introduction.

Definition 2: A (uniform) *continuous-time Markov chain (CTMC)* is a tuple $\mathcal{C} = (S, \mathbf{R})$ where S is as in Definition 1 and

- $\mathbf{R}: (S \times S) \rightarrow \mathbb{R}_{\geq 0}$ is the *rate matrix* such that there is a *uniformisation rate* $\mathbf{u}(\mathcal{C}) > 0$ with $\sum_{s' \in S} \mathbf{R}(s, s') = \mathbf{u}(\mathcal{C})$ for all $s \in S$.

If C is clear from context, we write \mathbf{u} instead of $\mathbf{u}(C)$. By $X^{C, s_0}: (\Omega_C \times \mathbb{R}_{\geq 0}) \rightarrow S$ with $s_0 \in S$ we denote the uniquely defined stochastic process [47] of C with initial state s_0 , where Ω_C is the sample space to be used.

In a *non-uniform CTMC*, the requirement $\sum_{s' \in S} \mathbf{R}(s, s') = \mathbf{u}(C)$ does not hold for all states. Every non-uniform CTMC can be transformed into an equivalent uniform CTMC with the same stochastic behaviour by increasing $\mathbf{R}(s, s')$ such that the total sum is the same for all states [47]. We require uniformity only for ease of presentation, it does not restrict the applicability of the methods developed here to general CTMCs.

The behaviour of a CTMC $C = (S, \mathbf{R})$ is similar to a DTMC. However, the durations until state changes are now real numbers. They are chosen according to independent negative exponential distributions with parameter \mathbf{u} . Thus, the probability that a state change takes place within time t is $1 - \exp(-\mathbf{u}t)$. The successor state is then selected according to the distribution $\mu: S \rightarrow [0, 1]$ with $\mu(s') = \frac{\mathbf{R}(s_0, s')}{\mathbf{u}}$ for all $s' \in S$. We assume that the process runs until a certain point of time \mathbf{t} is reached.

As for DTMCs, we assume that we have probability measures and expectations on the sample spaces.

We need to specify another discrete- and a continuous-time model [49], [50], which will later on be used to abstract large CTMCs. In addition to stochastic behaviour, these models also feature a *nondeterministic* choice over the successor distributions. Nondeterministic choices are choices which cannot be assigned a probability a priori. Instead, different stochastic behaviours result according to the resolution of the nondeterminism.

Definition 3: A *discrete-time Markov decision process (DTMDP)* is a tuple $\mathcal{D} = (S, \text{Act}, \mathbf{P})$ where S is as in Definition 1 and

- Act is a set of *actions*,
- $\mathbf{P}: (S \times \text{Act} \times S) \rightarrow [0, 1]$ is the *probability matrix* such that $\sum_{s' \in S} \mathbf{P}(s, \alpha, s') \in \{0, 1\}$ for all $s \in S$ and $\alpha \in \text{Act}$.

For $s \in S$, we denote the set of *enabled* actions with $\text{Act}(s) \stackrel{\text{def}}{=} \{\alpha \in \text{Act} \mid \sum_{s' \in S} \mathbf{P}(s, \alpha, s') = 1\}$. We require that either $|\text{Act}| < \infty$ or that for all $s \in S$ and all $p: S \rightarrow \mathbb{R}_{\geq 0}$ the set $\{\sum_{s' \in S} \mathbf{P}(s, \alpha, s') \cdot p(s') \mid \alpha \in \text{Act}(s)\}$ is compact.

The behaviour of a DTMDP is such that upon entering a state $s \in S$, an action $\alpha \in \text{Act}(s)$, or possibly a distribution over actions, is chosen. This choice determines the probabilities of the state the model moves to in the next time step. Notice that we do indeed allow uncountably many actions, with the given restriction.

Definition 4: A *(uniform) continuous-time Markov decision process (CTMDP)* is a tuple $C = (S, \text{Act}, \mathbf{R})$ where S and Act are as in Definition 3, $\mathbf{R}: (S \times \text{Act} \times S) \rightarrow \mathbb{R}_{\geq 0}$ is the *rate matrix* such that there is a fixed $\mathbf{u}(C)$ where for all $s \in S$ and $\alpha \in \text{Act}$ it is $\sum_{s' \in S} \mathbf{R}(s, \alpha, s') \in \{0, \mathbf{u}(C)\}$. If C is clear from the context, we write \mathbf{u} instead of $\mathbf{u}(C)$. For $s \in S$, we denote the set of *enabled* actions with $\text{Act}(s) \stackrel{\text{def}}{=} \{\alpha \in \text{Act} \mid \sum_{s' \in S} \mathbf{R}(s, \alpha, s') = \mathbf{u}\}$. We require that either $|\text{Act}| < \infty$ or that for all $s \in S$ and all $p: S \rightarrow \mathbb{R}_{\geq 0}$ the set $\{\sum_{s' \in S} \frac{\mathbf{R}(s, \alpha, s')}{\mathbf{u}} \cdot p(s') \mid \alpha \in \text{Act}(s)\}$ is compact.

As in a DTMDP, upon entering a state s , an action $\alpha \in \text{Act}(s)$ (or a distribution over this set) is chosen to determine the distribution over the successor states. As for CTMCs, the model

moves to this successor state after a time given according to the negative exponential distribution with parameter \mathbf{u} .

We need the following transformation from continuous-time to discrete-time models.

Definition 5: Given a CTMDP $C = (S, \text{Act}, \mathbf{R})$, the *embedded DTMDP* is defined as $\text{emb}(C) \stackrel{\text{def}}{=} (S, \text{Act}, \mathbf{P})$ with $\mathbf{P}(s, \alpha, s') \stackrel{\text{def}}{=} \frac{\mathbf{R}(s, \alpha, s')}{\mathbf{u}}$ for all $s, s' \in S$ and $\alpha \in \text{Act}(s)$.

We introduce a formalism to specify CTMDPs, extending the *abstract Markov chains* by Klink et al. [26], [28], [51], which is also a specific form of a *constraint Markov chain* [52]. The purpose of this model is to efficiently represent CTMDPs with a large number of actions. Instead of explicitly enumerating all possible choices over successor distributions, it allows to specify lower and upper bounds on the rates between states.

Definition 6: An *extended abstract continuous-time Markov chain (ECTMC)* is a tuple $C = (S, \widehat{\text{Act}}, \mathbf{I}^\ell, \mathbf{I}^u)$ where S is as in Definition 3 and $\widehat{\text{Act}}$ is a finite set. We consider the uniformisation rate $\mathbf{u}(C)$ of the model. The *intervals* are partial functions of the form $\mathbf{I}^\ell, \mathbf{I}^u: (S \times \widehat{\text{Act}}) \rightarrow (S \rightarrow [0, \mathbf{u}])$. We require \mathbf{I}^ℓ and \mathbf{I}^u to have the same domain. In addition, for each $s \in S$ there must be at least one $\hat{\alpha}$ such that $\mathbf{I}^\ell(s, \hat{\alpha})$ is defined.

The *CTMDP semantics* of an ECTMC is defined as $\llbracket C \rrbracket \stackrel{\text{def}}{=} (S, \text{Act}, \mathbf{R})$. We have $(\hat{\alpha}, v) \in \text{Act}$ if $\hat{\alpha} \in \widehat{\text{Act}}$ and if v is of the form $v: S \rightarrow [0, \mathbf{u}]$ with $\sum_{s \in S} v(s) = \mathbf{u}$. It is $(\hat{\alpha}, v) \in \text{Act}(s)$ if $\mathbf{I}^\ell(s, \hat{\alpha})$ and $\mathbf{I}^u(s, \hat{\alpha})$ are defined and for all $s' \in S$ it is $v(s') \in [\mathbf{I}^\ell(s, \hat{\alpha})(s'), \mathbf{I}^u(s, \hat{\alpha})(s')]$. We let $\mathbf{R}(s, (\hat{\alpha}, v), s') \stackrel{\text{def}}{=} v(s')$. An ECTMC thus represents a CTMDP in which for each state s one chooses a possible action $\hat{\alpha}$ of $\widehat{\text{Act}}$. In addition, one has to choose an assignment of successor rates which fulfil the requirement on the intervals. This way, the action set is uncountably large, but satisfies the requirements of Definition 4. The difference to the model of Klink et al. is the choice of $\hat{\alpha} \in \widehat{\text{Act}}$ before the choice of the successor rates. This allows to obtain more precise abstractions than we could obtain if we were using (non-extended) abstract Markov chains, while it still allows to implement efficient analysis methods, as seen later in Section 3 and Section 4.

To obtain a stochastic process from nondeterministic models, the nondeterminism must be resolved. *Schedulers* (or *policies*) formalise the mechanism to do so. Below, we define the most powerful class of schedulers we consider in this paper, and the stochastic processes they induce. A scheduler of this class can resolve the nondeterminism according to the states and actions (and their sequence) that were visited before the model moved to the current state. It may also decide not to pick one specific action, but rather involve a probabilistic choice over the enabled actions of a state. It is however neither aware of the exact time at which former events happened, nor of the current time.

Definition 7: A *time-abstract, history-dependent, randomised scheduler (HR)* for a DTMDP $\mathcal{D} = (S, \text{Act}, \mathbf{P})$ or a CTMDP $C = (S, \text{Act}, \mathbf{R})$ is a function $\sigma: ((S \times \text{Act})^* \times S) \rightarrow \text{Distr}(\text{Act})$ such that for all $\beta \in (S \times \text{Act})^*$ and $s \in S$ we have that if $\sigma(\beta, s)(\alpha) > 0$ then $\alpha \in \text{Act}(s)$. With Σ_{HR} we denote the set of all HRs.

Definition 8: Assume we are given a CTMDP $C = (S, \text{Act}, \mathbf{R})$ and a HR $\sigma: ((S \times \text{Act})^* \times S) \rightarrow \text{Distr}(\text{Act})$. We

define the *induced CTMC* as $C_\sigma \stackrel{\text{def}}{=} (S', \mathbf{R}')$ with

- $S' \stackrel{\text{def}}{=} (S \times \text{Act})^* \times S$,
- $\mathbf{R}'((\beta, s), (\beta, s, \alpha, s')) \stackrel{\text{def}}{=} \sigma(\beta, s)(\alpha) \cdot \mathbf{R}(s, \alpha, s')$ for $\beta \in (S \times \text{Act})^*$, $s, s' \in S$, $\alpha \in \text{Act}$, and $\mathbf{R}'(\cdot, \cdot) \stackrel{\text{def}}{=} 0$ else.

Let $X^{C_\sigma, s_0}: (\Omega_{C_\sigma} \times \mathbb{R}_{\geq 0}) \rightarrow ((S \times \text{Act})^* \times S)$ be the stochastic process of the CTMC C_σ with initial state $s_0 \in S$ and let $f: ((S \times \text{Act})^* \times S) \rightarrow S$ with $f(\beta, s) \stackrel{\text{def}}{=} s$. The *induced stochastic process* $X^{C, \sigma, s_0}: (\Omega_{C_\sigma} \times \mathbb{R}_{\geq 0}) \rightarrow S$ of C and σ starting in s_0 is then defined as $X_t^{C, \sigma, s_0} \stackrel{\text{def}}{=} f \circ X_t^{C_\sigma, s_0}$ for $t \in \mathbb{R}_{\geq 0}$. Definitions for DTMDPs are likewise using \mathbf{P} instead of \mathbf{R} .

We specify a simpler subclass of the schedulers of Definition 7. Schedulers of this class are only aware of the number of state changes that have happened so far, and may only choose a specific successor distribution rather than a distribution over them.

Definition 9: A *time-abstract, history-abstract, counting, deterministic scheduler (CD)* for a DTMDP $\mathcal{D} = (S, \text{Act}, \mathbf{P})$ or a CTMDP $C = (S, \text{Act}, \mathbf{R})$ is a function $\sigma: (S \times \mathbb{N}) \rightarrow \text{Act}$ such that for all $s \in S$ and $n \in \mathbb{N}$ if $\sigma(s, n) = \alpha$ then $\alpha \in \text{Act}(s)$. With Σ_{CD} we denote the set of all CDs.

Definition 10: Assume we are given a CTMDP $C = (S, \text{Act}, \mathbf{R})$ and a CD $\sigma: (S \times \mathbb{N}) \rightarrow \text{Act}$. We define the *induced CTMC* as $C_\sigma \stackrel{\text{def}}{=} (S', \mathbf{R}')$ with

- $S' \stackrel{\text{def}}{=} S \times \mathbb{N}$,
- $\mathbf{R}'((s, n), (s', n+1)) \stackrel{\text{def}}{=} \mathbf{R}(s, \sigma(s, n), s')$ for $s, s' \in S$ and $n \in \mathbb{N}$, and $\mathbf{R}'(\cdot, \cdot) \stackrel{\text{def}}{=} 0$ else.

Let $X^{C_\sigma, s_0}: (\Omega_{C_\sigma} \times \mathbb{R}_{\geq 0}) \rightarrow (S \times \mathbb{N})$ be the stochastic process of the CTMC C_σ and let $f: (S \times \mathbb{N}) \rightarrow S$ with $f(s, n) \stackrel{\text{def}}{=} s$. The *induced stochastic process* $X^{C, \sigma, s_0}: (\Omega_{C_\sigma} \times \mathbb{R}_{\geq 0}) \rightarrow S$ of C and σ starting in s_0 is then defined as $X_t^{C, \sigma, s_0} \stackrel{\text{def}}{=} f \circ X_t^{C_\sigma, s_0}$ for $t \in \mathbb{R}_{\geq 0}$. Definitions for DTMDPs are likewise using \mathbf{P} instead of \mathbf{R} .

We equip our models with *reward structures*, assigning values to states.

Definition 11: A *reward structure* for a stochastic process $X: (\Omega \times \mathbb{R}_{\geq 0}) \rightarrow S$ or a CTMC or CTMDP with state set S is a tuple $(\mathbf{r}_c, \mathbf{r}_f)$ with $\mathbf{r}_c: S \rightarrow \mathbb{R}_{\geq 0}$ and $\mathbf{r}_f: S \rightarrow \mathbb{R}_{\geq 0}$. We call \mathbf{r}_c the *cumulative reward rate* and \mathbf{r}_f the *final reward value*. We let $\mathbf{r}_f^{\max} \stackrel{\text{def}}{=} \max_{s \in S} \mathbf{r}_f(s)$ and $\mathbf{r}_c^{\max} \stackrel{\text{def}}{=} \max_{s \in S} \mathbf{r}_c(s)$.

For CTMCs, the cumulative reward rate $\mathbf{r}_c(s)$ is the reward obtained per time unit for staying in state s , until the time bound \mathbf{t} is reached. The final reward value $\mathbf{r}_f(s)$ specifies the reward one obtains for being in state s at time \mathbf{t} . We are interested in the expected values of these numbers, as formalised in Definition 12. For CTMDPs, we strive for the maximal (and analogously the minimal) value under all possible schedulers in the class we considered.

Definition 12: Given a time bound $\mathbf{t} \in \mathbb{R}_{\geq 0}$, the *value* of a stochastic process $X: (\Omega \times \mathbb{R}_{\geq 0}) \rightarrow S$ with a reward structure $\mathbf{r} = (\mathbf{r}_c, \mathbf{r}_f)$ is defined as $\mathbf{V}(X, \mathbf{r}, \mathbf{t}) \stackrel{\text{def}}{=} \mathbf{E}[\int_0^{\mathbf{t}} \mathbf{r}_c(X_u) du + \mathbf{r}_f(X_{\mathbf{t}})]$. For a CTMC $C = (S, \mathbf{R})$ and $s_0 \in S$, we let $\mathbf{V}(C, s_0, \mathbf{r}, \mathbf{t}) \stackrel{\text{def}}{=} \mathbf{V}(X^{C, s_0}, \mathbf{r}, \mathbf{t})$. For a CTMDP $C = (S, \text{Act}, \mathbf{R})$, the *maximal value* for $s_0 \in S$ is defined as $\mathbf{V}^{\max}(C, s_0, \mathbf{r}, \mathbf{t}) \stackrel{\text{def}}{=} \max_{\sigma \in \Sigma_{HR}} \mathbf{V}(X^{C, \sigma, s_0}, \mathbf{r}, \mathbf{t})$.

The interpretation of rewards and values depends on the model under consideration. For instance, in a CTMC representing a chemical reaction, we might assign a final reward value of n

to state s if s contains n molecules of a given species. This way, the value of the CTMC represents the expected number of this species at a given point of time.

We do not explicitly consider impulse (instantaneous) rewards $\mathbf{r}_i: (S \times S) \rightarrow \mathbb{R}_{\geq 0}$ for CTMCs here, that is rewards obtained for moving from one state to another. However, given cumulative reward rates \mathbf{r}_c and impulse rewards \mathbf{r}_i , we can define cumulative reward rates \mathbf{r}'_c as $\mathbf{r}'_c(s) \stackrel{\text{def}}{=} \mathbf{r}_c(s) + \sum_{s' \in S} \mathbf{R}(s, s') \cdot \mathbf{r}_i(s, s')$. For the properties under consideration, this new reward structure is equivalent to the one which uses impulse rewards (follows from [53, (6)]). Definition 12 resembles the approach considered in a recent paper [54], where we maximised the value over a more general class of schedulers than the one of Definition 7.

An important special case of the value is the *time-bounded reachability probability*, as it occurs for instance in the time-bounded until property of the probabilistic logic CSL [55]. Given a set of *target states* \mathbf{B} , we can express the probability to be in \mathbf{B} at time t by using a reward structure with $\mathbf{r}_c(s) = 0$ for all $s \in S$, and $\mathbf{r}_f(s) = 1$ if $s \in \mathbf{B}$ and $\mathbf{r}_f(s) = 0$ else. For the probability to reach \mathbf{B} within time t , we additionally modify the rate matrix \mathbf{R} such that $\mathbf{R}(s, s) = \mathbf{u}$ and $\mathbf{R}(s, s') = 0$ for $s' \neq s$ if $s \in \mathbf{B}$. For CTMCs, the case to reach \mathbf{B} within an interval $[a, b]$ with $0 < a < b$ can be handled by two successive analyses [55, Theorem 3]. The unbounded until $([0, \infty))$ and $[a, \infty))$ can be handled similarly [56, Section 4.4].

The fact that Definition 12 involves both final and cumulative rewards for CTMCs allows to express the values at different points of time in the following way. Assume we want to consider the cumulative reward rates v_1, v_2, \dots at consecutive time points $\mathbf{t}_1 = \delta_1, \mathbf{t}_2 = \mathbf{t}_1 + \delta_2, \dots$. A short calculation then shows that $v_1 = \mathbf{V}(C, s, (\mathbf{r}_c, 0), \delta_1)$, $v_2 = \mathbf{V}(C, s, (\mathbf{r}_c, v_1), \delta_2)$, ... The formulation for the final reward at different points of time is likewise.

PRISM [4] is a widely used tool, which features a guarded command language to model CTMCs (among other classes). For our purposes, it suffices to take a rather abstract view on the high-level modelling language used by this tool.

Definition 13: A *PRISM model (PM)* is a tuple $m = (\text{Var}, \text{init}, C, \text{succ}, R_c, R_f)$ where Var is a set of Boolean variables, with $\text{init}: \text{Var} \rightarrow \{0, 1\}$ we denote the *initial state* and C is a finite set of *commands*. Let $S_{\text{Var}} \stackrel{\text{def}}{=} \{s: \text{Var} \rightarrow \{0, 1\}\}$. The *cumulative reward rate* is a function $R_c: S_{\text{Var}} \rightarrow \mathbb{R}_{\geq 0}$ as is the *final reward value* $R_f: S_{\text{Var}} \rightarrow \mathbb{R}_{\geq 0}$. The *successor function* is a partial function of the form $\text{succ}: (S_{\text{Var}} \times C) \rightarrow (S_{\text{Var}} \times \mathbb{R}_{\geq 0})$. We define $\overline{\text{succ}}(s, c) \stackrel{\text{def}}{=} s'$ if $\text{succ}(s, c) = (s', \lambda)$ for some $\lambda \in \mathbb{R}_{\geq 0}$. We also let

$$\text{succ}(s) \stackrel{\text{def}}{=} \{(s', \lambda) \mid \exists c. \overline{\text{succ}}(s, c) = s'\} \\ \wedge \lambda = \sum_{\lambda': \exists c'. (s', \lambda') = \text{succ}(s, c')} \lambda'.$$

Further, $\overline{\text{succ}}(s) \stackrel{\text{def}}{=} \{s' \mid \exists \lambda. (s', \lambda) \in \text{succ}(s)\}$. Let $\overline{\text{succ}}^0 \stackrel{\text{def}}{=} \text{init}$ and $\overline{\text{succ}}^{i+1} \stackrel{\text{def}}{=} \{\overline{\text{succ}}(s) \mid s \in \overline{\text{succ}}^i\}$. The set of *reachable states (state space)* is $S_m \stackrel{\text{def}}{=} \bigcup_{i=0}^{\infty} \overline{\text{succ}}^i$. We require that there is $\mathbf{u}(m) > 0$ such that for all $s \in S_m$ it is $\mathbf{u}(m) = \sum \{\lambda \mid \exists s'. (s', \lambda) \in \text{succ}(s)\}$ (where the latter is a multiset).

The complete PRISM syntax also defines models consisting of several *modules*, that is, sets of guarded commands, which may synchronise or interleave. However, as the semantics of a model with several modules is defined as one with a single module, a single set of commands suffices. PRISM also allows to specify commands with several pairs of successors $(s'_1, \lambda_1), \dots, (s'_n, \lambda_n)$. For PMs describing CTMCs, such a command is equivalent to a set of n commands c_i in the above form: Each of them must be activated (defined) in the same states as the original command, and we then have $\text{succ}(c_i) = (s'_i, \lambda_i)$. The bounded integers PRISM supports can be represented by a binary encoding. Impulse rewards can be transformed to cumulative rewards, as discussed for CTMCs. For models in which there is no $\mathbf{u}(m)$ with the required property, we can add an extra command to increase the self-loop rate where necessary.

The formal semantics of a PM is as follows.

Definition 14: Consider a PM $m = (\text{Var}, \text{init}, C, \text{succ}, R_c, R_f)$. The *induced CTMC* is $C_m \stackrel{\text{def}}{=} (S_m, \mathbf{R})$ such that for all $s, s' \in S_m$ we have $\mathbf{R}(s, s') \stackrel{\text{def}}{=} \lambda$ if $(s', \lambda) \in \text{succ}(s)$ and $\mathbf{R}(s, s') \stackrel{\text{def}}{=} 0$ if no such tuple exists. The *induced reward structure* is $\mathbf{r}_m \stackrel{\text{def}}{=} (R_c, R_f)$.

In Section 3 we will abstract CTMCs into ECTMCs. To do so, we will subsume several concrete states of a CTMC to abstract states of an ECTMC.

Definition 15: Given a PM m , a *partitioning* of the state space S_m is a finite ordered set $\mathfrak{P} = \langle \mathfrak{z}_0, \dots, \mathfrak{z}_{n-1} \rangle$ of non-empty, pairwise disjoint subsets of S_m such that $S_m = \bigcup_{i=0}^{n-1} \mathfrak{z}_i$.

Binary decision diagrams [57] are an efficient tool to symbolically represent structures which are too large to be represented in an explicit form.

Definition 16: We fix a finite ordered set $V \stackrel{\text{def}}{=} \langle x_1, \dots, x_m \rangle$ of *Boolean variables*. A *binary decision diagram* (BDD) is a rooted acyclic directed Graph b with node set N and root node n_{root} . There are two types of nodes in N : *outer nodes* n , which do not have out-going edges and which are labelled with a value $v(n) \in \{0, 1\}$. The remaining nodes are *inner nodes* $n \in N$, which have exactly two successor nodes, denoted by $h(n)$ and $l(n)$. Inner nodes n are labelled with a variable $v(n) \in V$.

A *variable valuation* is a function $v: V \rightarrow \{0, 1\}$. We denote the set of all variable valuations by Val . Each valuation v induces a unique path in the BDD from the root node to an outer node: At an inner node n we follow the edge to $h(n)$ if $v(v(n)) = 1$ and the edge to $l(n)$ if $v(v(n)) = 0$. The function $\llbracket b \rrbracket: \text{Val} \rightarrow \{0, 1\}$ represented by a BDD b returns for a variable valuation v the value of the outer node reached by following the path induced by v .

Definition 17: A BDD is *ordered* if for all inner nodes n the following condition holds: Either $h(n)$ is an outer node or $v(n) < v(h(n))$ and the same for $l(n)$. A BDD is *reduced* if all sub-BDDs rooted at the different nodes of the BDD represent distinct functions. Reduced and ordered BDDs are called *OBDDs*.

The OBDD for the constant 0 (1) function, which consists of a single outer node labelled with 0 (1, resp.), is denoted in the sequel by bdd_0 (bdd_1 , resp.).

OBDDs are a canonical representation (up to isomorphism) of arbitrary functions $f: \text{Val} \rightarrow \{0, 1\}$ [57]. In the following

we will only use OBDDs. For more details on (O)BDDs we refer the reader to [57], [58].

OBDDs support a wide number of operations like the Boolean operations \wedge , \vee , and \neg . Given two ordered sets $V_1 = \langle x_{i_1}, \dots, x_{i_n} \rangle$ and $V_2 = \langle x_{j_1}, \dots, x_{j_m} \rangle$ of Boolean variables, by $b' = b[V_1/V_2]$ we denote the OBDD which results from renaming the variables in V_1 to the corresponding variables in V_2 . For $V' \subseteq V$ and OBDD b , we let $\llbracket \exists V'. b \rrbracket(v) = \bigvee \{ \llbracket b \rrbracket(v') \mid \forall x \notin V'. v'(x) = v(x) \}$ be the existential quantification of the variables in V' .

We can use OBDDs to represent PMs in a symbolic form, if we leave out the stochastic aspects.

Definition 18: Consider a PM $m = (\text{Var}, \text{init}, C, \text{succ}, R_c, R_f)$. The *OBDD representation* of m is a tuple $\mathbf{b}_m \stackrel{\text{def}}{=} (\text{Var}, \text{Var}', \text{init}, \{\text{succ}_c\}_{c \in C})$. There Var and Var' are sets of Boolean variables with $\text{Var} \cap \text{Var}' = \emptyset$ such that there is a one-to-one mapping between variables $x \in \text{Var}$ and $x' \in \text{Var}'$. Further, init and succ_c are OBDDs over the variables Var and $\text{Var} \cup \text{Var}'$, respectively. We require that $\llbracket \text{init} \rrbracket(v) = 1$ iff for all $x \in \text{Var}$ it is $\text{init}(x) = v(x)$. For all succ_c we require $\llbracket \text{succ}_c \rrbracket(v) = 1$ iff for all $x \in \text{Var}$ it is $v(x) = s(x)$, $v(x') = s'(x)$ and $\overline{\text{succ}}(s, c) = s'$. By succ we denote the OBDD such that $\llbracket \text{succ} \rrbracket = \bigvee_{c \in C} \llbracket \text{succ}_c \rrbracket$.

OBDDs can also be used to symbolically represent a partitioning of the state space of a PM. Let $\mathfrak{P} = \langle \mathfrak{z}_0, \dots, \mathfrak{z}_{n-1} \rangle$ be a partitioning of the PM $m = (\text{Var}, \text{init}, C, \text{succ}, R_c, R_f)$. The idea is to assign to each block \mathfrak{z}_i of \mathfrak{P} a unique block number i and to use a binary representation of i , which is encoded using $k = \lceil \log_2 n \rceil$ novel BDD variables $\mathfrak{B} = \langle x_0, \dots, x_{k-1} \rangle$.

Definition 19: The *OBDD representation* of $\mathfrak{P} = \langle \mathfrak{z}_0, \dots, \mathfrak{z}_{n-1} \rangle$ is the OBDD $\mathbf{b}_{\mathfrak{P}}$ over the variables $\mathfrak{B} \uplus \text{Var}$, where $\mathfrak{B} = \langle x_0, \dots, x_{k-1} \rangle$ with $k = \lceil \log_2 n \rceil$. We require that $\llbracket \mathbf{b}_{\mathfrak{P}} \rrbracket(v) = 1$ iff there is $s \in S_m$ such that for all $x \in \text{Var}$ we have $v(x) = s(x)$ and there is $\mathfrak{z} \in \mathfrak{P}$ such that $s \in \mathfrak{z}$ and for all $x \in \mathfrak{B}$ we have $v(x) = \mathfrak{z}(x)$. With $\mathfrak{z}_i(x_j) \stackrel{\text{def}}{=} (i \div 2^j) \bmod 2$ we denote the value of variable $x_j \in \mathfrak{B}$ in the binary encoding of the block number i of $\mathfrak{z}_i \in \mathfrak{P}$. With $\mathbf{b}_{\mathfrak{z}}$ we denote the OBDD such that $\llbracket \mathbf{b}_{\mathfrak{z}} \rrbracket(v) = 1$ iff v represents a state of \mathfrak{z} .

Regarding the variable order of $\text{Var} \uplus \mathfrak{B}$, we assume in the following that all variables in Var precede all variables in \mathfrak{B} . This leads to more efficient algorithms for accessing the block number of a given state.

3 ALGORITHMS

In this section, we first describe an algorithm to approximate minimal and maximal values of CTMDPs. Afterwards, we describe how to obtain an ECTMC from a PM, such that its induced CTMDP is a valid abstraction (cf. Proposition 2) of the CTMC semantics of the PM. We provide an algorithm which computes an ECTMC overapproximation of a PM given in an OBDD representation. Using the first algorithm, we can obtain intervals from this abstraction which are guaranteed to contain the actual value of the CTMC.

3.1 Computing Reward Values for CTMDPs

Let $\phi_\lambda(i) \stackrel{\text{def}}{=} \frac{\lambda^i}{i!} e^{-\lambda}$ denote the probabilities of a Poisson distribution with parameter λ , and let $\psi_\lambda(i) \stackrel{\text{def}}{=} \sum_{j=i+1}^{\infty} \phi_\lambda(j) = 1 - \sum_{j=0}^i \phi_\lambda(j)$.

The algorithm to compute the maximal values of CTMDPs is given in Algorithm 1. The input is a CTMDP C with reward structure $\mathbf{r} = (\mathbf{r}_c, \mathbf{r}_f)$, and the precision $\varepsilon > 0$ up to which the values are to be computed. The algorithm for the minimum is likewise, replacing max by min. The requirement on the

Algorithm 1: Compute maximal values for $C = (S, Act, \mathbf{R})$, $\mathbf{r} = (\mathbf{r}_c, \mathbf{r}_f)$ up to ε .

```

1 let  $k$  s.t.  $\sum_{i=0}^k \psi_{\text{ut}}(i) > \mathbf{ut} - \frac{\varepsilon \mathbf{u}}{2\mathbf{r}_c^{\max}} \wedge \psi_{\text{ut}}(k) \cdot \mathbf{r}_f^{\max} < \frac{\varepsilon}{2}$ 
2  $C' = (S, Act, \mathbf{P}) := \text{emb}(C)$ 
3 forall  $s \in S$  do  $q_{k+1}(s) := 0$ 
4 forall  $i = k, k-1, \dots, 0$  do
5   forall  $s \in S$  do
6      $m := \max_{\alpha \in Act(s)} \sum_{s' \in S} \mathbf{P}(s, \alpha, s') q_{i+1}(s')$ 
7      $q_i(s) := m + \phi_{\text{ut}}(i) \cdot \mathbf{r}_f(s) + \psi_{\text{ut}}(i) \cdot \frac{\mathbf{r}_c(s)}{\mathbf{u}}$ 
8 return  $q_0$ 

```

actions in Definition 4 assures that the maximum in Algorithm 1 exists. We can also directly apply this algorithm on ECTMCs without constructing the uncountably large induced CTMDPs. The crucial part here is the optimisation over the uncountable actions, which can be done using a slight adaption of methods from [26, Chapter 4.1]. There, optimising the assignment of successor rates with restrictions given by lower and upper bounds is already described. Because of this, for each $s \in S$ we can apply the method described there for each \hat{a} such that $\mathbf{I}^\ell(s, \hat{a})$ (and thus \mathbf{I}^u) is defined, thus to find the optimal $v_{\hat{a}}: S \rightarrow [0, \mathbf{u}]$ for this \hat{a} . Afterwards, we choose the optimal $v_{\hat{a}}: S \rightarrow [0, \mathbf{u}]$ among all \hat{a} , which is easy as there are only finitely many.

Proposition 1: Let $C = (S, Act, \mathbf{R})$ be a CTMDP with reward structure $\mathbf{r} = (\mathbf{r}_c, \mathbf{r}_f)$. Then there exists $\sigma \in \Sigma_{CD}$ such that $\mathbf{V}^{\max}(C, s_0, \mathbf{r}, \mathbf{t}) = \mathbf{V}(X^{C, \sigma, s_0}, \mathbf{r}, \mathbf{t})$ for all $s_0 \in S$. Further, the return value q_0 of Algorithm 1 is such that $|\mathbf{V}^{\max}(C, s_0, \mathbf{r}, \mathbf{t}) - q_0(s_0)| < \varepsilon$.

Proof sketch: At first, we show that we can simulate each history-dependent randomised scheduler by a randomised counting scheduler (CR). In contrast to CD, these schedulers may be randomised, but, as CDs, only know the number of steps which have passed rather than the full history. For this, we use a result about discrete-time Markov chains. Next, we show that Algorithm 1 cannot yield values which are larger than the maximal value resulting from such a CR. Then, we show that the algorithm does not return values which are larger than the value obtained by any CR plus the specified precision.

Looking at the decisions the algorithm takes at Line 6, we can reconstruct a prefix of the decisions of a CD. By letting the precision approach 0, we can show that there is indeed a complete CD yielding the same value. \square

The full proof can be found in Appendix A.

Algorithm 1 generalises an approach from a previous paper about time-bounded reachability [39] using results by Kwiatkowska et al. [53]. Its correctness also proves that deterministic counting schedulers suffice to obtain optimal values, because the algorithm implicitly computes such a scheduler.

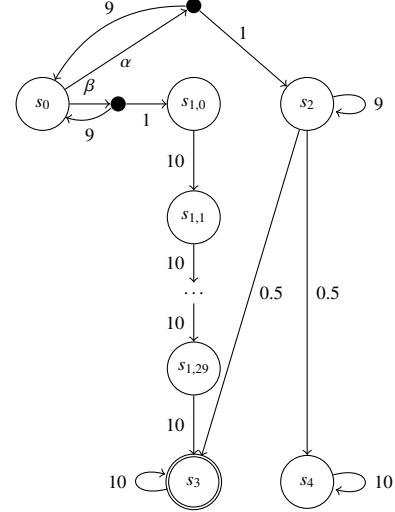


Fig. 1. Example to show that we do not compute exact extremal interval-bounded until probabilities of CTMDPs.

It is also related to an earlier work in queueing theory [59], which is however different in a number of ways. The target there was to obtain approximations for a more general class of schedulers of CTMDPs than we need here, and thus does not consider maxima over HRs explicitly. It assumes a fixed maximal number of steps to happen in the uniformised DTMDP, rather than deriving the necessary number, as we do in our algorithm. [59] is also more involved with models featuring a particular structure rather computing conservative bounds on properties of CTMCs.

The next proposition states how CTMDPs can be used to overapproximate CTMCs.

Proposition 2: Let $C = (S, \mathbf{R})$ be a CTMC with reward structure $(\mathbf{r}_c, \mathbf{r}_f)$ and let $\mathfrak{P} = \langle \mathfrak{z}_0, \dots, \mathfrak{z}_{n-1} \rangle$ be a partitioning of S . Consider the CTMDP $C' \stackrel{\text{def}}{=} (\mathfrak{P}, Act, \mathbf{R}')$ where for each $\mathfrak{z} \in \mathfrak{P}$ and $s \in \mathfrak{z}$ we find $\alpha_s \in Act$ such that for all $\mathfrak{z}' \in \mathfrak{P}$ we have $\mathbf{R}'(\mathfrak{z}, \alpha_s, \mathfrak{z}') \stackrel{\text{def}}{=} \sum_{s' \in \mathfrak{z}'} \mathbf{R}(s, s')$. Further, consider a reward structure $(\mathbf{r}'_c, \mathbf{r}'_f)$ such that for all $\mathfrak{z} \in \mathfrak{P}$ it is $\mathbf{r}'_c(\mathfrak{z}) \geq \max_{s \in \mathfrak{z}} \mathbf{r}_c(s)$ and $\mathbf{r}'_f(\mathfrak{z}) \geq \max_{s \in \mathfrak{z}} \mathbf{r}_f(s)$. Then for all $\mathfrak{z}_0 \in \mathfrak{P}$ and $s_0 \in \mathfrak{z}_0$ we have $\mathbf{V}(C, s_0, \mathbf{r}, \mathbf{t}) \leq \mathbf{V}^{\max}(C', \mathfrak{z}_0, \mathbf{r}', \mathbf{t})$.

Proof sketch: We can construct a scheduler σ such that the embedded DTMDP of C' mimics the behaviour of the embedded DTMC of C , so that in each step the probability to be in a given abstract state \mathfrak{z} is the sum of the probabilities of being in a state s of C with $s \in \mathfrak{z}$. By the definition of reward structures, the value obtained in C' using σ is at least as high as the value in C . As the maximal value in C' is at least as high as the one using σ , the result follows. \square

The full proof can be found in Appendix B.

As mentioned before, CSL bounded-until properties can be expressed using rewards. Their probabilities can thus also be bounded using Algorithm 1. Notice, however, that for the case of intervals $[a, b]$ or $[a, \infty)$, the successive application of the algorithm only bounds the value in the CTMC that has been abstracted. It does not guarantee to yield the extremal until probabilities of the CTMDP over HRs in these cases; we only compute a bound for these values.

Example 1: Consider the CTMDP C in Figure 1, which is adapted from previous publications [60], [54]. The only difference is that previously non-uniform CTMDPs were used, that is CTMDPs in which the sum of leaving rates may be different for each state. Here, we have increased the self-loop rates thus to obtain a uniformisation rate of $\mathbf{u} = 10$. States of the model are given as circles, in which the state name is written. If there is more than one activated action in a state, we draw all of them as small black circles connected to the corresponding states. Non-zero transition rates are then drawn starting in the corresponding black circle. In case there is just one possible action in a state, we directly draw the transitions from this state. Rates with value 0 are left out.

We assign a cumulative reward rate of 0 to each state. The final reward is 0 for all states except for s_3 , where it is 1. Intuitively, if we want to maximise the value, the optimal choice of the action depends on the amount of time left. If much time is left, it is best to choose β in s_0 , because this action leads to a sequence of states which, given an infinite amount of time, always reaches s_3 . If little time is left, it is better to choose α , because then s_3 can be reached quickly, although there is a significant chance that this state will not be reached at all.

With this model and reward structure, we can compute the probabilities of the CSL formula $\mathcal{P}(\text{true } \mathcal{U}^{[0,4]} s_3)$ by computing the value for $\mathbf{t} = 4$. Because the state s_3 is absorbing, this value is also the probability of the interval-bounded until property $\mathcal{P}(\text{true } \mathcal{U}^{[1,4]} s_3)$.

In this special case, we can thus compute this probability using the method developed in this paper or by the previous algorithm by Baier et al. [39]. For s_0 , this value is $\mathbf{V}^{\max}(C, s_0, (0, \mathbf{r}_f), 4) \approx 0.659593$.

In case this CTMDP is the abstraction of a given CTMC, we can apply two consecutive analyses to bound the interval-bounded until probability. For this, we first compute $v(\cdot) \stackrel{\text{def}}{=} \mathbf{V}^{\max}(C, s_0, (0, \mathbf{r}_f), \cdot, 3)$, and afterwards we consider $v'(\cdot) \stackrel{\text{def}}{=} \mathbf{V}^{\max}(C, s_0, (0, v), \cdot, 1)$. We now have $v'(s_0) \approx 0.671162$, which shows that this value is an upper bound for the until probability in the original model. It is indeed between the value obtained using HRs as discussed previously and the one obtained using time-dependent schedulers [60], [54]. This value is larger than the one considered in the last paragraph, which shows that a consecutive analysis does not yield the maximum interval-bounded reachability probability over all HRs in a given CTMDP. The reason that this happens is that by dividing the analysis into two parts, the schedulers of the two analyses can change their decisions more often and obtain more information than they are supposed to have. Instead of only having the information that the objective is to optimise the reward until $\mathbf{t} = 4$, it is now also known whether $\mathbf{t} \leq 1$ or not.

From the discussion of the values of consecutive time points $\mathbf{t}_1 = \delta_1, \mathbf{t}_2 = \mathbf{t}_1 + \delta_2, \dots$, it follows that we can also use the algorithm to compute bounds for them in an efficient way. Instead of doing analyses with time bounds $\mathbf{t}_1, \mathbf{t}_2, \dots$, we only have to do analyses with values $\delta_1, \delta_2, \dots$, which might be much lower than $\mathbf{t}_1, \mathbf{t}_2, \dots$. Thus, the fact that the algorithm allows to handle final and cumulative rewards at the same time has the potential to speed up such a series of analyses.

3.2 Abstracting PRISM Models

To take advantage of Proposition 2, we want to avoid to actually construct the CTMC to be abstracted. Doing this allows us to handle models which are too large to be handled in an explicit-state form. For this, we can use non-probabilistic model checkers which feature a guarded-command language, like NuSMV [61]. Such a tool can work with an OBDD-based representation of PMs as in Definition 18, and compute the set of reachable states. We can then specify some OBDDs representing *predicates*, i.e., sets of concrete states. These can be used to split the state space, by subsuming all concrete states that are contained in the same subset of predicates, thus to obtain an OBDD partitioning as in Definition 19.

Next, we consider the ECTMC abstraction of a given PRISM model.

Definition 20: Consider a PM $m = (\text{Var}, \text{init}, C, \text{succ}, R_c, R_f)$ with induced CTMC $C = (S, \mathbf{R})$ and a partitioning $\mathfrak{P} = \langle \mathfrak{z}_0, \dots, \mathfrak{z}_{n-1} \rangle$ of its state space. The *ECTMC abstraction* of m is defined as $C \stackrel{\text{def}}{=} (\mathfrak{P}, \widehat{\text{Act}}, \mathbf{I}^\ell, \mathbf{I}^u)$ with $\widehat{\text{Act}} \stackrel{\text{def}}{=} \{\hat{a} : C \rightarrow \mathfrak{P}\}$. We let $A(\mathfrak{z}, \hat{a})$ denote the set of all $s \in \mathfrak{z}$ such that $\text{Dom}(\text{succ}(s, \cdot)) = \text{Dom}(\hat{a})$ (that is, the domains of the two partial functions agree) and for all applicable $c \in C$ we have $\text{succ}(s, c) \in \hat{a}(c)$. We then choose the domain of \mathbf{I}^ℓ and \mathbf{I}^u such that for all $\mathfrak{z} \in \mathfrak{P}$ it is

$$\text{Dom}(\mathbf{I}^\ell(\mathfrak{z}, \cdot)) \stackrel{\text{def}}{=} \text{Dom}(\mathbf{I}^u(\mathfrak{z}, \cdot)) \stackrel{\text{def}}{=} \{\hat{a} \in \widehat{\text{Act}} \mid A(\mathfrak{z}, \hat{a}) \neq \emptyset\}.$$

Then for $\mathfrak{z}, \mathfrak{z}' \in \mathfrak{P}$ and $\hat{a} \in \text{Dom}(\mathbf{I}^u(\mathfrak{z}, \cdot))$ we define

$$\mathbf{I}^\ell(\mathfrak{z}, \hat{a})(\mathfrak{z}') \stackrel{\text{def}}{=} \min_{s \in A(\mathfrak{z}, \hat{a})} \sum_{\substack{(s', c) \in \text{succ}(s), \\ s' \in \mathfrak{z}'}} \lambda,$$

and accordingly \mathbf{I}^u using max. The abstract reward structure $\mathbf{r} \stackrel{\text{def}}{=} (\mathbf{r}_c, \mathbf{r}_f)$ is defined as $\mathbf{r}_c(\mathfrak{z}) \stackrel{\text{def}}{=} \max_{s \in \mathfrak{z}} R_c(s)$ and $\mathbf{r}_f(\mathfrak{z}) \stackrel{\text{def}}{=} \max_{s \in \mathfrak{z}} R_f(s)$.

By construction, the CTMDP semantics of the ECTMC fulfils the requirements of Proposition 2 for a correct abstraction of the CTMC semantics of the PRISM model. It is also monotone in the sense that, by using a refined partitioning, we cannot obtain worse bounds than with a coarser partitioning.

Proposition 3: Consider a PM $m = (\text{Var}, \text{init}, C, \text{succ}, R_c, R_f)$ with a partitioning $\mathfrak{P} = \langle \mathfrak{z}_0, \dots, \mathfrak{z}_{n-1} \rangle$ of the state space of its induced CTMC, and a further partitioning $\mathfrak{P}' = \langle \mathfrak{z}'_0, \dots, \mathfrak{z}'_{n-1} \rangle$ such that for each $\mathfrak{z}_t \in \mathfrak{P}$ we find $\mathfrak{z}'_{t,1}, \dots, \mathfrak{z}'_{t,u} \in \mathfrak{P}'$ such that $\mathfrak{z}_t = \bigcup_{j=1}^u \mathfrak{z}'_{t,j}$.

Then for two ECTMC abstractions $C \stackrel{\text{def}}{=} (\mathfrak{P}, \widehat{\text{Act}}, \mathbf{I}^\ell, \mathbf{I}^u)$ and $C' \stackrel{\text{def}}{=} (\mathfrak{P}', \widehat{\text{Act}}, \mathbf{I}^{\ell'}, \mathbf{I}^{u'})$ with corresponding reward structures \mathbf{r} and \mathbf{r}' we have $\mathbf{V}^{\max}(C, \mathfrak{z}_t, \mathbf{r}, \mathbf{t}) \geq \mathbf{V}^{\max}(C', \mathfrak{z}'_{t,j}, \mathbf{r}', \mathbf{t})$.

Proof sketch: We can show that for an arbitrary $\varepsilon > 0$, we have that \mathbf{V}^{\max} of a state \mathfrak{z} of the original partition plus ε is at least equal to the value of a state \mathfrak{z}' of the refined partition for which we have $\mathfrak{z}' \subseteq \mathfrak{z}$. This implies that the same holds for $\varepsilon = 0$, which means that the value of a state of the refined partition cannot be higher than the one of the original abstraction.

We use the fact that we can apply Algorithm 1 to compute values up to any precision $\varepsilon > 0$. Consider the runs of the algorithm on the original and refined partitioning. Before the execution of the main loop at Line 4, we have that $q_{k+1}(\mathfrak{z}) = q_{k+1}(\mathfrak{z}') = 0$. For each execution of the main loop, we have

a maximising decision in \mathfrak{z}' , leading to $v_i(\mathfrak{z}')$ to be added to $q_{i+1}(\mathfrak{z}')$ to obtain $q_i(\mathfrak{z}')$. We can construct a decision for \mathfrak{z} such that $v_i(\mathfrak{z}') \leq v_i(\mathfrak{z})$. This means that in each iteration of the main loop, the q_i of \mathfrak{z}' can never become larger than the one of \mathfrak{z} . Thus, after the termination of the main loop and the algorithm, the value obtained for the coarser abstraction is at least as high as the one in the refined abstraction. \square

The full proof can be found in Appendix C.

With a given partitioning, we can apply Algorithm 2 to obtain an abstraction of the model. The algorithm computes an ECTMC to provide an upper bound of the model value; a corresponding algorithm for the lower bound can be defined likewise.

The algorithm does some initialisations and afterwards, in line 5, calls Algorithm 3. This algorithm descends into the OBDD partitioning (lines 3 to 14), visiting each state of the model explicitly. When a specific state s contained in an abstract state \mathfrak{z} is reached (lines 16 to 29), we extend the abstracting model to take into account the behaviour of this state. In lines 17 and 18, we extend the upper bounds for the reward rates of \mathfrak{z} such that they are at least as high as those of s . Notice that to compute the reward rates of this state we use the original high-level PM, not the OBDD representation. Then, in lines 20 to 29 we handle the transition rates, thus to include the rates of the concrete state. We again use the high-level model, this time to compute the set of commands that are enabled in the current state (line 20), the corresponding action \hat{a} (line 21, cf. Definition 20), and the concrete successor states with their corresponding rates (line 22). For each of them we use the function **sAbs** to obtain the abstract state it belongs to. For all successor states we add up the rates to the same abstract state (line 26). Then, starting from line 27, we apply the actual widening of the rates.

Function **sAbs** works as follows: Since we use a variable order in which the variables encoding states are placed above the variables for the abstract states, each state $s \in S$ induces a path in the OBDD P which ends at the OBDD node that represents the abstract state of s . We follow the unique path, given by the encoding of s to the outer nodes labelled with 1. This yields the encoding of the abstract state of s . The runtime of **sAbs** is therefore linear in the number of OBDD variables.

Let $n = |S|$ be the number of concrete states of the model under consideration, let k be the total number of positive transitions and let c be the number of OBDD variables. Algorithm 3 visits each state and transition once. Since the block number variables are placed at the bottom of the variable order, accessing the block number of a state in function **sAbs** has a runtime of $O(c)$. From this, we have that the overall complexity of Algorithm 2 is $O((n + k) \cdot c)$.

In the discussion so far, we assumed that it is already clear how the set of concrete states shall be divided into abstract states. We might however come across models where this is not clear, or where the results obtained from the abstraction are unsatisfactory. In these cases, we have to apply *refinement*, that is, split existing abstract states into new ones. For other model types, such refinement procedures already exist [31], [32]. In the analysis types considered before, schedulers sufficed which fix a decision per state, and take neither the past history nor

Algorithm 2: Compute ECTMC and reward structure from a given partitioning \mathfrak{P} with OBDD $b_{\mathfrak{P}} = (N, n_{\mathfrak{P}}, h, l, v)$ of PM $m = (Var, init, C, succ, R_c, R_f)$.

```

1  $\mathbf{I}^\ell(\cdot, \cdot)(\cdot) := \text{undefined}$ 
2  $\mathbf{I}^u(\cdot, \cdot)(\cdot) := \text{undefined}$ 
3  $\mathbf{r}_c(\cdot) := \mathbf{r}_f(\cdot) := -\infty$ 
4  $\mathfrak{A} := \{0, 1, \dots, |\mathfrak{P}| - 1\}$ 
5  $\text{approx}(n_{\mathfrak{P}}, 0)$ 
6 return  $(\mathfrak{A}, \mathbf{I}^\ell, \mathbf{I}^u), (\mathbf{r}_c, \mathbf{r}_f)$ 
```

Algorithm 3: Procedure $\text{approx}(n, \text{level})$.

```

1 if  $n = \text{bdd}_0$  then return
2 else if  $\text{level} < \text{leafLevel}$  then
3   // We are still at a variable level.
4    $x = \text{varAtLevel}(\text{level})$ 
5   if  $n \neq \text{bdd}_1$  and  $x = v(n)$  then
6      $n_l := l(n), n_h := h(n)$ 
7   else
8      $n_l := n, n_h := n$ 
9   if  $x \in \mathfrak{V}$  then
10      $\mathfrak{z}(x) := 0, \text{approx}(n_l, \text{level} + 1)$ 
11      $\mathfrak{z}(x) := 1, \text{approx}(n_h, \text{level} + 1)$ 
12   else
13      $s(x) := 0, \text{approx}(n_l, \text{level} + 1)$ 
14      $s(x) := 1, \text{approx}(n_h, \text{level} + 1)$ 
15 else
16   // We have traversed all variable levels.
17    $\mathbf{r}_c(\mathfrak{z}) := \max(\mathbf{r}_c(\mathfrak{z}), R_c(s))$ 
18    $\mathbf{r}_f(\mathfrak{z}) := \max(\mathbf{r}_f(\mathfrak{z}), R_f(s))$ 
19
20    $C := \text{Dom}(\text{succ}(s, \cdot))$  // commands enabled in s
21    $\hat{a} = \{(c, \text{sAbs}(n_{\mathfrak{P}}, s')) \mid c \in C \wedge s' = \overline{\text{succ}}(s, c)\}$ 
22    $A := \text{succ}(s)$ 
23    $\Lambda(\cdot) := 0$ 
24   forall  $(s', \lambda) \in A$  do
25      $\mathfrak{z}' := \text{sAbs}(n_{\mathfrak{P}}, s')$ 
26      $\Lambda(\mathfrak{z}') := \Lambda(\mathfrak{z}') + \lambda$ 
27   forall  $\mathfrak{z}' \in \mathfrak{A}$  do
28      $\mathbf{I}^\ell(\mathfrak{z}, \hat{a})(\mathfrak{z}') := \min(\mathbf{I}^\ell(\mathfrak{z}, \hat{a})(\mathfrak{z}'), \Lambda(\mathfrak{z}'))$ 
29      $\mathbf{I}^u(\mathfrak{z}, \hat{a})(\mathfrak{z}') := \max(\mathbf{I}^u(\mathfrak{z}, \hat{a})(\mathfrak{z}'), \Lambda(\mathfrak{z}'))$ 
```

number of steps before the state was entered into account. Then, depending on the decisions of the scheduler per state, new predicates are introduced to split the state space. In our case, such simple schedulers are not sufficient to obtain extremal values, as has already been shown for the simpler case of time-bounded reachability [39]. Thus, it is not clear how to introduce predicates to split the state space.

As a first heuristic, we do the following: We treat an OBDD representation of a PM as a labelled transition system, in which the commands play the role of the labels. We then use an existing algorithm to symbolically compute (non-probabilistic) strong bisimulations [62], but stop the algorithm after a number of steps. This way, we obtain a partitioning in the form of

Definition 19. As we will see later in Section 4, although the method is not guaranteed to yield a good abstraction, it can work well in practice.

The method discussed works for a very general class of PMs and arbitrary state partitionings. However, because it is based on explicitly visiting each concrete state at least once, it may take much time to perform for large models. To tackle this problem, a parallel implementation of the technique is possible. Given a computing system with a number of processors, one can symbolically divide the states of the model, such that each processor works on a different part of the OBDD representing the state space. Each processor can then process the model part it is assigned to. The only point of interaction is the widening of the rates and reward rates of the abstract model. On a shared memory architecture, one could use different semaphores for the reward rates and successor transitions of each abstract state to avoid delays. Without shared memory, the processors can compute partial abstractions independently, which are merged after the computations are finished. This technique is faster, but has the disadvantage of having to store several (partial) copies of the abstraction. If the state space is divided such that all states of an abstract state are assigned to a single processor, no locking is needed and the overhead is reduced.

As an alternative to parallelisation, it should also be possible to use optimisation methods over BDDs [63], [64], [65], [10] to compute the rate and reward intervals symbolically rather than rely on explicit enumeration of all possible variable assignments.

4 CASE STUDIES

To show the practicality of the method, we applied it on two case studies from classical performance and dependability engineering [22], [23]. We implemented the techniques of Algorithm 1 and Algorithm 2. To represent the ECTMCs, we used a sparse-matrix-like data structure.

Where possible, we compared the results to PRISM. PRISM always starts by building an MTBDD representation of the model under consideration. The subsequent analysis is then performed using *value iteration* in the CTMC semantics similarly to Algorithm 1. The data structure used here is either an MTBDD, a sparse matrix, or a *hybrid* structure [33]. In the latter, values for the model states are stored explicitly, but parts of the transition structure are stored implicitly.

For all experiments we used a Quad-Core AMD Opteron™ Processor 8356 (of which we only used one core) with 2300 MHz and 64 GB of main memory.

4.0.0.1 Workstation cluster [22].: We consider a fault-tolerant workstation cluster. It consists in two sub-clusters, which, in turn, contain N workstations connected via a central switch. The two switches are connected via a backbone. Each component of the system can break down, and is then fixed by a single repair unit responsible for the entire system.

We are interested in the expected number of repairs until a time bound of $t = 500$, a property which can be expressed using cumulative reward rates. For N up to 512, the model has been successfully analysed before using PRISM¹. While the existing

analysis methods worked well for model instantiations up to this N and somewhat above, the techniques do not work well anymore for a very large number of workstations. Constructing the model using MTBDDs seems not to be problematic, but the subsequent analyses cannot be performed successfully. The sparse-matrix and the hybrid method fail at some point, because they rely on an explicit representation of the state space and thus run out of memory. Also the MTBDD-based value iteration fails at some point, and works rather slowly. The reason is probably that during the value iteration a large number of different non-terminal nodes appear, which make the MTBDD complex and thus large and slow to operate on. Detailed information about the performance of PRISM on this case study is given in Table 1. By $|S|$ and $|R|$ we give the approximate number of states and transitions, resp., of the original CTMC model. For each of the three PRISM engines we give the runtime (columns “Time”) and memory consumption (columns “Memory”) for computing the expected rewards. An entry “– Time out –” means that PRISM did not terminate within 160,000 seconds, an entry “– Memory out –” that more than 60 GB of memory are required to complete the analysis.

In Table 2 we apply the method developed in this paper on several instantiations of the number of workstations N . The results we obtained by our method are given in “ECTMC Results”. Besides the runtime and memory consumption, we give in the column titled “ $|\mathfrak{P}|$ ” the number of abstract states we used for the corresponding analysis. The column “Interval” gives the lower and upper bounds of the actual value of the expected reward.

As we see from the time and memory usage, for smaller models it is advantageous to use an explicit-state method as implemented in PRISM, because of the additional overhead our method introduces. As instantiations become larger, using the method of this paper becomes worthwhile. While we do not always get precise bounds for all analyses performed with this number of abstract states, we always were able to compute the order of magnitude. Interestingly, the value bounds get tighter with an increasing number of model states.

As discussed in Section 3, we apply a heuristical refinement algorithm based on bisimulations for labelled transition systems. We use the symbolic algorithm [62] for computing (non-stochastic) strong bisimulations to obtain a suitable state partitioning. We abort its fix-point iteration prematurely after a user-specified number n of iterations. In Figure 2 we show how the quality of the approximation evolves depending on n . The behaviour of the cluster case study is shown on the left. One can observe that the width of the computed interval converges quickly to the actual value when increasing the number of iterations.

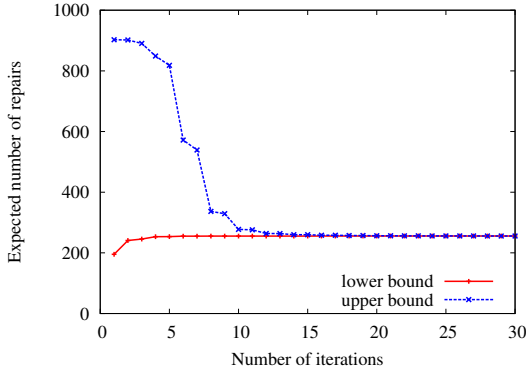
Notably, if we use the same number of refinement steps, for all model instantiations considered, $|\mathfrak{P}|$ stayed constant, although the number of model states $|S|$ was different for each instantiation (cf. Table 2).

Table 3 contains more detailed running times for the cluster benchmark with $N = 2048$ workstations using the ECTMC abstraction for different numbers of bisimulation iterations, which are given in the first column. The second column contains the number of abstract states. The running times in seconds are

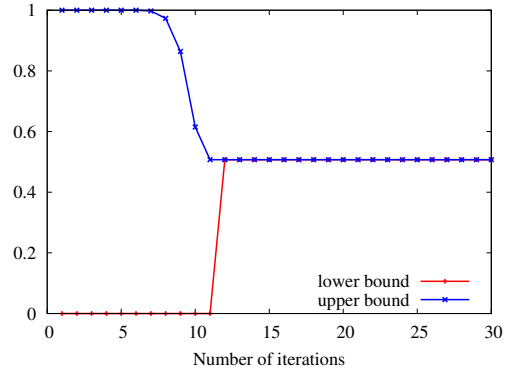
1. <http://www.prismmodelchecker.org/casestudies/cluster.php#mc>, Property $R\{\text{"num_repairs"}\}=?[C \leq T]$.

TABLE 1
PRISM results for the number of repairs in the workstation cluster until $t = 500$

N	$ S $	$ R $	sparse engine		hybrid engine		symbolic engine		Result
			Time	Memory	Time	Memory	Time	Memory	
32	$3.87 \cdot 10^4$	$1.86 \cdot 10^5$	9.29	36.67	14.90	37.48	13 791.20	184.49	64.17635
64	$1.51 \cdot 10^5$	$7.33 \cdot 10^5$	62.11	42.92	88.93	41.69	– Time out –	–	127.98101
128	$5.97 \cdot 10^5$	$2.91 \cdot 10^6$	380.51	60.18	585.55	54.48	– Time out –	–	255.48297
256	$2.37 \cdot 10^6$	$1.16 \cdot 10^7$	3 182.73	141.71	4 737.73	98.27	– Time out –	–	509.58417
512	$9.47 \cdot 10^6$	$4.62 \cdot 10^7$	10 540.54	817.39	14 965.74	284.66	– Time out –	–	896.80612
1 024	$3.78 \cdot 10^7$	$1.85 \cdot 10^8$	13 242.08	3 154.91	25 513.31	1 014.79	– Time out –	–	905.19921
2 048	$1.51 \cdot 10^8$	$7.39 \cdot 10^8$	– Time out –	–	– Time out –	–	– Time out –	–	??
4 096	$6.04 \cdot 10^8$	$2.95 \cdot 10^9$	– Memory out –	–	– Time out –	–	– Time out –	–	??
8 192	$2.42 \cdot 10^9$	$1.18 \cdot 10^{10}$	– Memory out –	–	– Memory out –	–	– Time out –	–	??
16 384	$9.66 \cdot 10^9$	$4.72 \cdot 10^{10}$	– Memory out –	–	– Memory out –	–	– Time out –	–	??



(a) Workstation cluster with $N = 128$



(b) Google file system with $M = 256$

Fig. 2. Quality of the ECTMC approximation for different numbers of bisimulation iterations

TABLE 2
Number of repairs in the workstation cluster until time $t = 500$.

N	$ R $	ECTMC Results		
		Time	Memory	Interval
32	19 420	107.15	90.03	[64.176, 64.199]
64	19 420	109.43	86.28	[127.980, 128.490]
128	19 420	115.93	89.54	[255.455, 259.797]
256	19 420	132.89	94.58	[509.000, 580.485]
512	19 420	181.99	91.87	[869.749, 900.052]
1 024	19 420	412.43	107.22	[905.018, 905.200]
2 048	19 420	1 335.54	103.31	[905.766, 905.767]
4 096	19 420	5 298.29	104.89	[905.955, 905.955]
8 192	19 420	28 361.36	132.48	[906.040, 906.040]
16 384	19 420	147 691.30	139.56	[906.084, 906.084]

given separately for the different main operations: The call to NuSMV to generate the OBDDs for the underlying transition system (col. 3), the given number of bisimulation iterations (col. 4), the construction of the ECTMC from the partitioning (col. 5), the value iteration to compute the reward interval (col. 6) and finally the total computation time (col. 7). The last two columns contain the memory consumption in Megabytes and the computed reward interval.

4.0.0.2 Google file system [23], [66].: We additionally consider a replicated file system as used as part of the Google search engine. Originally, the model was given as a generalised stochastic Petri net, but was transformed to a PM for the analysis.

Files are divided into *chunks* of equal size. Several copies of

each chunk reside at several *chunk servers*. There is a single *master server* which knows the location of the chunk copies. If a user of the file system wants to access a certain chunk of a file, it asks the master for the location. Data transfer then takes place directly between a chunk server and the user. The model describes the life cycle of a single chunk, but accounts for the load caused by the other chunks.

The model features three parameters: M is the number of chunk servers, with C we denote the number of chunks a chunk server may store, and the total number of chunks is N .

We consider the minimal probability over all states in which severe hardware problems have occurred (master server is down and more than three quarters of the chunk servers are down), that within time t a state will be reached in which a guaranteed quality-of-service level (all three chunk copies are present and the master server is available) holds. This is a bounded-reachability property and thus based on final rewards.

We fix $C = 5000$, $N = 100\,000$ and $t = 60$ and provide results for several M in Table 5. In the analyses with PRISM (see Table 4), we used an improved OBDD variable order, such that the performance results are better than in a previous publication [66]. In contrast to the previous case study, PRISM's sparse matrix engine was faster and did not use more memory compared to the hybrid engine. The symbolic engine was again the slowest. The MTBDD representation of the model requires more memory per concrete state compared to the previous case study. We assume that this is because the number of different rates occurring is much higher, and because some of the rates

TABLE 3

Detailed experimental results for the workstation cluster ($N = 2048$, $t = 500$) using the ECTMC abstraction

Iterations	$ P $	Running Times					Memory	Interval
		NuSMV	Refinement	ECTMC	Value Iter.	Total		
5	2440	64.75	0.93	991.16	13.45	1070.34	56.13	[902.092, 905.771]
10	9216	68.46	6.96	1135.57	41.68	1252.79	70.11	[905.739, 905.767]
15	19420	65.25	18.26	1029.23	116.89	1229.78	103.31	[905.766, 905.767]
20	34596	73.93	43.71	1150.70	213.72	1482.29	134.62	[905.767, 905.767]
25	52600	69.82	78.88	1070.13	321.43	1540.49	180.85	[905.767, 905.767]
30	76176	66.43	126.85	1052.87	466.31	1712.76	227.80	[905.767, 905.767]

TABLE 4

PRISM results for the Google file system

N	$ S $	$ R $	sparse engine		hybrid engine		symbolic engine		Result
			Time	Memory	Time	Memory	Time	Memory	
32	$6.15 \cdot 10^3$	$4.03 \cdot 10^4$	1.60	607.42	1.68	607.78	67.95	624.01	0.000000
64	$2.46 \cdot 10^4$	$1.66 \cdot 10^5$	34.72	614.24	71.04	616.84	66319.89	791.23	0.507119
128	$9.83 \cdot 10^4$	$6.77 \cdot 10^5$	464.25	624.81	1 890.41	627.23	– Time out –	–	0.507119
256	$3.93 \cdot 10^5$	$2.74 \cdot 10^6$	3 083.24	683.27	26 132.52	701.11	– Time out –	–	0.507119
512	$1.57 \cdot 10^6$	$1.11 \cdot 10^7$	41 901.82	938.10	– Time out –	–	– Time out –	–	0.507119
1 024	$6.29 \cdot 10^6$	$4.44 \cdot 10^7$	– Time out –	–	– Time out –	–	– Time out –	–	??
2 048	$2.52 \cdot 10^7$	$1.78 \cdot 10^8$	– Time out –	–	– Time out –	–	– Time out –	–	??

TABLE 5

Google file system with property 12 and $t = 60$, $N = 100\,000$, $C = 5000$.

M	$ P $	ECTMC Results		
		Time	Mem.	Interval
32	6 138	30.28	72.28	[0.0000, 0.0000]
64	18 042	251.86	86.53	[0.5071, 0.5071]
128	29 515	822.08	142.23	[0.5071, 0.5071]
256	29 515	1 531.46	147.66	[0.5071, 0.5071]
512	29 515	2 951.37	129.72	[0.5071, 0.5071]
1 024	29 515	6 776.42	128.40	[0.5071, 0.5071]
2 048	29 515	14 957.94	137.47	[0.5071, 0.5071]

are obtained by multiplying state variables, thus leading to a more complex MTBDD structure. Notice that in this model, from a certain value of M onward, the probability discussed is almost the same.

We give detailed information on the instance with $M = 128$ of the Google file system in Table 6 and Figure 2 (right-hand side) for different numbers n of bisimulation iterations. We can again observe that the computed interval for the bounded-reachability property quickly converges to the actual probability with increasing n .

5 CONCLUSION

We developed an efficient method to compute extremal values of CTMDPs over HR schedulers. It can be used to safely bound quantities of interest of CTMCs, by abstracting them into a special class of CTMDPs, and then applying this method. Experimental results have shown that the approach works well in practice.

There are a number of possible future works: The current refinement technique surely does not yield an optimal partitioning of the state space in all cases. We thus want to see how the scheduler we implicitly obtain by Algorithm 1 can be used to refine the model. The abstraction technique could also be extended to other property classes and models. For

instance, models already involving nondeterminism could be abstracted and approximated using Markov games [67], [68], [69]. It would also be interesting to see how a parallelised or symbolic abstraction method sketched at the end of Section 3 performs against the one currently implemented. Using a three-valued logic [26], the technique could also be integrated into an existing probabilistic (CSL) model checker.

5.0.0.3 Acknowledgements.: We thank Martin Neuhäuser and Lijun Zhang for fruitful discussions.

REFERENCES

- [1] B. Köpf and D. A. Basin, “Automatically deriving information-theoretic bounds for adaptive side-channel attacks,” *Journal of Computer Security*, vol. 19, no. 1, pp. 1–31, 2011.
- [2] W. Sanders and J. F. Meyer, “Performability evaluation of distributed systems using stochastic activity networks,” in *Proc. of the 2nd International Workshop on Petri Nets and Performance Models*. IEEE Computer Society, 1987, pp. 111–120.
- [3] B. R. Haverkort, *Performance of computer communication systems – a model-based approach*. Wiley, 1998.
- [4] M. Kwiatkowska, G. Norman, and D. Parker, “PRISM 4.0: Verification of probabilistic real-time systems,” in *Proc. of CAV*, ser. Lecture Notes in Computer Science, vol. 6806. Springer-Verlag, 2011, pp. 585–591.
- [5] J.-P. Katoen, I. S. Zapreev, E. M. Hahn, H. Hermanns, and D. N. Jansen, “The ins and outs of the probabilistic model checker MRMC,” *Performance Evaluation*, vol. 68, no. 2, pp. 90–104, 2011.
- [6] G. Ciardo, A. S. Miner, and M. Wan, “Advanced features in SMART: the stochastic model checking analyzer for reliability and timing,” *SIGMETRICS Performance Evaluation Review*, vol. 36, no. 4, pp. 58–63, 2009.
- [7] D. D. Deavours, G. Clark, T. Courtney, D. Daly, S. Derisavi, J. M. Doyle, W. H. Sanders, and P. G. Webster, “The Möbius framework and its implementation,” *IEEE Transactions on Software Engineering*, vol. 28, no. 10, pp. 956–969, 2002.
- [8] D. Parker, “Implementation of symbolic model checking for probabilistic systems,” Ph.D. dissertation, University of Birmingham, UK, 2002.
- [9] H. Hermanns, M. Z. Kwiatkowska, G. Norman, D. Parker, and M. Siegle, “On the use of MTBDDs for performability analysis and verification of stochastic systems,” *Journal of Logic and Algebraic Programming*, vol. 56, no. 1–2, pp. 23–67, 2003.

TABLE 6
Detailed experimental results for the Google file system ($M = 128$, $t = 60$) using the ECTMC abstraction

Iterations	\mathcal{B}	Running Times					Memory	Interval
		NuSMV	Refinement	ECTMC	Value Iter.	Total		
5	3204	0.86	3.19	1.80	70.42	76.98	65.00	[0.000000, 0.999999]
10	15564	0.81	15.44	2.01	426.32	445.42	92.08	[0.000000, 0.614823]
15	29515	0.83	41.24	2.20	716.67	761.89	142.23	[0.507119, 0.507119]
20	42725	0.83	82.54	2.41	1040.74	1127.53	177.59	[0.507119, 0.507119]
25	57472	0.79	137.50	2.57	1349.73	1491.65	249.21	[0.507119, 0.507119]
30	69932	0.82	191.83	2.62	1652.66	1849.04	279.13	[0.507119, 0.507119]

- [10] M. Wan, G. Ciardo, and A. S. Miner, "Approximate steady-state analysis of large Markov models based on the structure of their decision diagram encoding," *Performance Evaluation*, vol. 68, no. 5, pp. 463–486, 2011.
- [11] K. Lampka, M. Siegle, J. Ossowski, and C. Baier, "Partially-shared zero-suppressed multi-terminal BDDs: concept, algorithms and applications," *Formal Methods in System Design*, vol. 36, no. 3, pp. 198–222, 2010.
- [12] W. H. Sanders, "Assuring the trustworthiness of the smarter electric grid," in *Proc. of NCA*. IEEE Computer Society, 2012.
- [13] A. Hartmanns, P. Berrang, and H. Hermanns, "A comparative analysis of decentralized power grid stabilization strategies," in *Winter Simulation Conference (WSC)*, 2012, (to appear).
- [14] M. Massink, D. Latella, A. Bracciali, M. D. Harrison, and J. Hillston, "Scalable context-dependent analysis of emergency egress models," *Formal Aspects of Computing*, vol. 24, no. 2, pp. 267–302, 2012.
- [15] M. Massink, D. Latella, A. Bracciali, and J. Hillston, "Modelling non-linear crowd dynamics in Bio-PEPA," in *Proc. of FASE*, ser. Lecture Notes in Computer Science, vol. 6603. Springer-Verlag, 2011, pp. 96–110.
- [16] S. Xu, W. Lu, and Z. Zhan, "A stochastic model of multivirus dynamics," *IEEE Trans. Dependable Sec. Comput.*, vol. 9, no. 1, pp. 30–45, 2012.
- [17] G. Zehender, E. Ebranati, F. Bernini, A. L. Presti, G. Rezza, M. Delogu, M. Galli, and M. Ciccozzi, "Phylogeography and epidemiological history of west nile virus genotype 1a in europe and the mediterranean basin," *Infection, Genetics and Evolution*, vol. 11, no. 3, pp. 646–653, Apr. 2011.
- [18] Y. Yuan and J. A. L. "Stochastic models for virus and immune system dynamics," *Mathematical Biosciences*, vol. 234, no. 2, pp. 84–94, Dec. 2011.
- [19] T. A. Henzinger, B. Jobstmann, and V. Wolf, "Formalisms for specifying Markovian population models," *Int'l Journal of Foundations of Computer Science*, vol. 22, no. 4, pp. 823–841, 2011.
- [20] J. Hillston, M. Tribastone, and S. Gilmore, "Stochastic process algebras: From individuals to populations," *Computer Journal*, vol. 55, no. 7, pp. 866–881, 2012.
- [21] M. Mateescu, V. Wolf, F. Didier, and T. A. Henzinger, "Fast adaptive uniformisation of the chemical master equation," *IET Syst Biol*, vol. 4, no. 6, pp. 441–52, 2010.
- [22] B. R. Haverkort, H. Hermanns, and J.-P. Katoen, "On the use of model checking techniques for dependability evaluation," in *Proc. of SRDS*. IEEE Computer Society, 2000, pp. 228–237.
- [23] L. Cloth and B. R. Haverkort, "Model checking for survivability," in *Proc. of QEST*. IEEE Computer Society, 2005, pp. 145–154.
- [24] R. Wimmer, B. Braitling, B. Becker, E. M. Hahn, P. Crouzen, H. Hermanns, A. Dhama, and O. E. Theel, "Symblicit calculation of long-run averages for concurrent probabilistic systems," in *Proc. of QEST*. IEEE Computer Society, 2010, pp. 27–36.
- [25] P. Crouzen, E. M. Hahn, H. Hermanns, A. Dhama, O. E. Theel, R. Wimmer, B. Braitling, and B. Becker, "Bounded fairness for probabilistic distributed algorithms," in *Proc. of ACSD*. IEEE Computer Society, 2011, pp. 89–97.
- [26] D. Klink, "Three-valued abstraction for stochastic systems," Ph.D. dissertation, RWTH Aachen, Germany, 2010.
- [27] M. J. A. Smith, "Compositional abstraction of PEPA models for transient analysis," in *Proc. of EPEW*, ser. Lecture Notes in Computer Science, vol. 6342. Springer-Verlag, 2010, pp. 252–267.
- [28] J.-P. Katoen, D. Klink, M. Leucker, and V. Wolf, "Three-valued abstraction for continuous-time Markov chains," in *Proc. of CAV*, ser. Lecture Notes in Computer Science, vol. 4590. Springer-Verlag, 2007, pp. 311–324.
- [29] P. Buchholz, "Bounding reward measures of Markov models using the Markov decision processes," *Numerical Linear Algebra with Applications*, vol. 18, no. 6, pp. 919–930, 2011.
- [30] L. de Alfaro and P. Roy, "Magnifying-lens abstraction for Markov decision processes," in *Proc. of CAV*, ser. Lecture Notes in Computer Science, vol. 4590. Springer-Verlag, 2007, pp. 325–338.
- [31] M. Kattenbelt, M. Z. Kwiatkowska, G. Norman, and D. Parker, "A game-based abstraction-refinement framework for Markov decision processes," *Formal Methods in System Design*, vol. 36, no. 3, pp. 246–280, 2010.
- [32] H. Hermanns, B. Wachter, and L. Zhang, "Probabilistic CEGAR," in *Proc. of CAV*, ser. Lecture Notes in Computer Science. Springer-Verlag, 2008, pp. 162–175.
- [33] M. Z. Kwiatkowska, G. Norman, and D. Parker, "Probabilistic symbolic model checking with PRISM: a hybrid approach," *Software Tools for Technology Transfer*, vol. 6, no. 2, pp. 128–142, 2004.
- [34] T. Dayar, *Analyzing Markov chains using Kronecker products. Theory and applications*. Springer-Verlag, 2012.
- [35] P. Buchholz, G. Ciardo, S. Donatelli, and P. Kemper, "Complexity of memory-efficient kronecker operations with applications to the solution of markov models," *INFORMS Journal on Computing*, vol. 12, no. 3, pp. 203–222, 2000.
- [36] A. Benoit, B. Plateau, and W. J. Stewart, "Memory-efficient kronecker algorithms with applications to the modelling of parallel systems," *Future Generation Comp. Syst.*, vol. 22, no. 7, pp. 838–847, 2006.
- [37] P. Fernandes, B. Plateau, and W. J. Stewart, "Efficient descriptor-vector multiplications in stochastic automata networks," *J. ACM*, vol. 45, no. 3, pp. 381–414, 1998.
- [38] P. Buchholz and P. Kemper, "Kronecker based matrix representations for large markov models," in *Validation of Stochastic Systems*, 2004, pp. 256–295.
- [39] C. Baier, H. Hermanns, J.-P. Katoen, and B. R. Haverkort, "Efficient computation of time-bounded reachability probabilities in uniform continuous-time Markov decision processes," *Theoretical Computer Science*, vol. 345, no. 1, pp. 2–26, 2005.
- [40] M. L. Puterman, *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley and Sons, 1994.
- [41] P. R. D'Argenio, B. Jeannot, H. E. Jensen, and K. G. Larsen, "Reachability analysis of probabilistic systems by successive refinements," in *Proc. of PAPM-PROBMIV*, ser. Lecture Notes in Computer Science, vol. 2165. Springer-Verlag, 2001, pp. 39–56.
- [42] W. K. Grassmann, "Finding Transient Solutions in Markovian Event Systems Through Randomization," in *Numerical Solution of Markov Chains*, 1991, pp. 357–371.
- [43] A. P. A. van Moorsel and W. H. Sanders, "Adaptive Uniformization," *Communications in Statistics - Stochastic Models*, vol. 10, no. 3, pp. 619–647, 1994.
- [44] B. Munsky and M. Khammash, "The finite state projection algorithm for the solution of the chemical master equation," *Journal of Chemical Physics*, vol. 124, no. 044104, 2006.
- [45] T. A. Henzinger, M. Mateescu, and V. Wolf, "Sliding window abstraction for infinite markov chains," in *CAV*, 2009, pp. 337–352.
- [46] P. Buchholz, "Finite horizon analysis of infinite ctmdps," in *DSN*, 2012, pp. 1–12.
- [47] W. J. Stewart, *Introduction to the Numerical Solution of Markov Chains*. Princeton University Press, 1994.
- [48] J. Kemeny, J. Snell, and A. Knapp, *Denumerable Markov Chains*. D. Van Nostrand Company, 1966.
- [49] R. A. Howard, *Dynamic Programming and Markov Processes*. John

Wiley and Sons, Inc., 1960.

- [50] D. P. Bertsekas, *Dynamic Programming and Optimal Control*. Athena Scientific, 2005.
- [51] I. Kozine and L. V. Utkin, “Interval-valued finite Markov chains,” *Reliable Computing*, vol. 8, no. 2, pp. 97–113, 2002.
- [52] B. Caillaud, B. Delahaye, K. G. Larsen, A. Legay, M. L. Pedersen, and A. Wasowski, “Compositional design methodology with constraint Markov chains,” in *Proc. of QEST*. IEEE Computer Society, 2010, pp. 123–132.
- [53] M. Kwiatkowska, G. Norman, and A. Pacheco, “Model checking expected time and expected reward formulae with random time bounds,” *Computers & Mathematics with Applications*, vol. 51, pp. 305–316, 2006.
- [54] P. Buchholz, E. M. Hahn, H. Hermanns, and L. Zhang, “Model checking algorithms for CTMDPs,” in *Proc. of CAV*, ser. Lecture Notes in Computer Science, vol. 6806. Springer-Verlag, 2011, pp. 225–242.
- [55] C. Baier, B. R. Haverkort, H. Hermanns, and J.-P. Katoen, “Model-checking algorithms for continuous-time Markov chains,” *IEEE Transactions on Software Engineering*, vol. 29, no. 6, pp. 524–541, 2003.
- [56] M. Z. Kwiatkowska, G. Norman, and D. Parker, “Stochastic model checking,” in *SFM*, ser. Lecture Notes in Computer Science, vol. 4486. Springer-Verlag, 2007, pp. 220–270.
- [57] R. E. Bryant, “Graph-based algorithms for Boolean function manipulation,” *IEEE Transactions on Computers*, vol. 35, no. 8, pp. 677–691, 1986.
- [58] I. Wegener, *Branching Programs and Binary Decision Diagrams – Theory and Applications*, ser. SIAM Monographs on Discrete Mathematics and Applications. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics (SIAM), 2000.
- [59] S. A. Lippman, “Countable-state, continuous-time dynamic programming with structure,” *Operations Research*, vol. 24, no. 3, pp. 477–490, 1976.
- [60] L. Zhang and M. R. Neuhäuser, “Model checking interactive Markov chains,” in *Proc. of TACAS*, ser. LNCS, vol. 6015. Springer, 2010, pp. 53–68.
- [61] A. Cimatti, E. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella, “NuSMV version 2: An opensource tool for symbolic model checking,” in *Proc. of CAV*, ser. Lecture Notes in Computer Science, vol. 2404. Springer-Verlag, 2002, pp. 359–364.
- [62] R. Wimmer, M. Herbstritt, H. Hermanns, K. Strampp, and B. Becker, “Sigref – A symbolic bisimulation model checker,” in *Proc. of ATVA*, ser. Lecture Notes in Computer Science, vol. 4218. Springer-Verlag, 2006, pp. 477–492.
- [63] Y.-T. Lai, M. Pedram, and S. B. K. Vrudhula, “EVBDD-based algorithms for integer linear programming, spectral transformation, and function decomposition,” *IEEE Transactions on CAD of Integrated Circuits and Systems*, vol. 13, no. 8, pp. 959–975, 1994.
- [64] D. E. Knuth, *The Art of Computer Programming, Volume 4, Fascicle 1: Bitwise Tricks & Techniques; Binary Decision Diagrams*, 12th ed. Addison-Wesley Professional, 2009.
- [65] J. Ossowski and C. Baier, “Symbolic reasoning with weighted and normalized decision diagrams,” *Electronic Notes in Theoretical Computer Science*, vol. 151, no. 1, pp. 39–56, 2006.
- [66] C. Baier, E. M. Hahn, B. R. Haverkort, H. Hermanns, and J.-P. Katoen, “Model checking for performability,” *Mathematical Structures in Computer Science*, 2012, (to appear).
- [67] T. Brázdil, V. Forejt, J. Krcál, J. Kretínský, and A. Kucera, “Continuous-time stochastic games with time-bounded reachability,” in *Proc. of FSTTCS*, ser. LIPIcs, vol. 4, 2009, pp. 61–72.
- [68] M. Rabe and S. Schewe, “Optimal time-abstract schedulers for CTMDPs and Markov games,” in *Proc. of QAPL*, ser. EPTCS, vol. 28, 2010, pp. 144–158.
- [69] M. N. Rabe and S. Schewe, “Finite optimal control for time-bounded reachability in CTMDPs and continuous-time Markov games,” *Acta Informatica*, vol. 48, no. 5-6, pp. 291–315, 2011.
- [70] R. Strauch, “Negative dynamic programming,” *Annals of Mathematical Statistics*, vol. 37, pp. 871–890, 1966.

APPENDIX A

PROOF OF PROPOSITION 1

We need some additional notations and lemmata to complete the proofs of the propositions in the paper.

Definition 21: Let $\mathcal{D} = (S, \mathbf{P})$ be a DTMC. For $s_0, s_k \in S$ and $k \in \mathbb{N}$ we define $\pi^{\mathcal{D}}(s_0, k, s_k) = \Pr(X_k^{\mathcal{D}, s_0} = s_k)$.

Definition 21 specifies the transient probability to be in state s_k at step k when having started in state s_0 .

Corollary 1: Notice that for $\mathcal{D} = (S, \mathbf{P})$, $k \in \mathbb{N}$ and $s_0, s_k \in S$ it is

$$\begin{aligned} & \pi^{\mathcal{D}}(s_0, k, s_k) \\ &= \mathbf{P}^k(s_0, s) \\ &\stackrel{\text{def}}{=} \sum_{s_1 \in S} \mathbf{P}(s_0, s_1) \cdot \sum_{s_2 \in S} \mathbf{P}(s_1, s_2) \cdot \sum_{s_3 \in S} \mathbf{P}(s_2, s_3) \\ &\quad \dots \sum_{s_{k-1}} \mathbf{P}(s_{k-2}, s_{k-1}) \cdot \mathbf{P}(s_{k-1}, s_k), \end{aligned}$$

that is, the transient probability in a DTMC can be expressed using matrix multiplications.

We extend the definition of values to discrete-time models, which will be used in the further parts of the proof.

Definition 22: Let $\mathcal{D} = (S, \mathbf{P})$ be a DTMC, let $\mathbf{r} = (\mathbf{r}_c, \mathbf{r}_f)$ be a reward structure and let $\mathbf{u}, \mathbf{t} \geq 0$. We define

$$\begin{aligned} & \mathbf{V}(\mathcal{D}, s_0, \mathbf{r}, \mathbf{t}, \mathbf{u}) \\ &\stackrel{\text{def}}{=} \sum_{i=0}^{\infty} \left(\phi_{\mathbf{u}\mathbf{t}}(i) \sum_{s_i \in S} \pi^{\mathcal{D}}(s_0, i, s_i) \mathbf{r}_f(s_i) \right. \\ &\quad \left. + \psi_{\mathbf{u}\mathbf{t}}(i) \sum_{s_i \in S} \pi^{\mathcal{D}}(s_0, i, s_i) \frac{\mathbf{r}_c(s_i)}{\mathbf{u}} \right). \end{aligned}$$

We define a type of schedulers which are simpler than the HRs of Definition 7 and at the same time generalise the CD of Definition 9.

Definition 23: A *time-abstract, history-abstract, counting, randomised scheduler (CR)* for a DTMDP $\mathcal{D} = (S, \text{Act}, \mathbf{P})$ or a CTMDP $C = (S, \text{Act}, \mathbf{R})$ is a function $\sigma: (S \times \mathbb{N}) \rightarrow \text{Distr}(\text{Act})$ such that for all $s \in S$ and $n \in \mathbb{N}$, if $\sigma(s, n)(\alpha) > 0$ then $\alpha \in \text{Act}(s)$. With Σ_{CR} we denote the set of all CRs.

Definition 24: Assume we are given a CTMDP $C = (S, \text{Act}, \mathbf{R})$ and a CR $\sigma: (S \times \mathbb{N}) \rightarrow \text{Distr}(\text{Act})$. We define the *induced CTMC* as $C_\sigma \stackrel{\text{def}}{=} (S', \mathbf{R}')$ with

- $S' \stackrel{\text{def}}{=} S \times \mathbb{N}$,
- $\mathbf{R}'((s, n), (s', n+1)) \stackrel{\text{def}}{=} \sum_{\alpha \in \text{Act}} \sigma(s, n)(\alpha) \cdot \mathbf{R}(s, \alpha, s')$ for $s, s' \in S$ and $n \in \mathbb{N}$, and $\mathbf{R}'(\cdot) \stackrel{\text{def}}{=} 0$ else.

Let $X^{C_\sigma, s_0}: (\Omega_{C_\sigma} \times \mathbb{R}_{\geq 0}) \rightarrow (S \times \mathbb{N})$ be the stochastic process of the CTMC C_σ and let $f: (S \times \mathbb{N}) \rightarrow S$ with $f(s, n) = s$. Induced DTMCs of DTMDPs are defined accordingly. The *induced stochastic process* $X^{C, \sigma, s_0}: (\Omega_{C_\sigma} \times \mathbb{R}_{\geq 0}) \rightarrow S$ of C and σ starting in $s_0 \in S$ is then defined as $X_t^{C, \sigma, s_0} = f \circ X_t^{C_\sigma, s_0}$ for $t \in \mathbb{R}_{\geq 0}$. Definitions for DTMDPs are likewise using \mathbf{P} instead of \mathbf{R} .

We extend the notation of transient probabilities to scheduled nondeterministic models. It is known that for DTMDPs CR schedulers are as powerful as HR schedulers.

Definition 25: For a DTMDP $\mathcal{D} = (S, \text{Act}, \mathbf{P})$ and a scheduler $\sigma \in \Sigma_{HR} \cup \Sigma_{CD} \cup \Sigma_{CR}$ we define $\pi^{\mathcal{D}, \sigma}(s_0, k, s_k) = \Pr(X_k^{\mathcal{D}, \sigma, s_0} = s_k)$ for all $k \in \mathbb{N}$ and $s_0, s_k \in S$.

Lemma 1: Consider a DTMDP $\mathcal{D} = (S, \text{Act}, \mathbf{P})$ and a HR σ_{hr} . Then there is a CR σ_{cr} such that for all $s_0, s_n \in S$ and $n \in \mathbb{N}$ we have $\pi^{\mathcal{D}, \sigma_{hr}}(s_0, n, s_n) = \pi^{\mathcal{D}, \sigma_{cr}}(s_0, n, s_n)$.

Proof: The proof is given in [70] and [40, Theorem 5.5.1] where CR are denoted as MR (Markov randomized) policies. \square

Definition 26: For a CTMC $C = (S, \mathbf{R})$, we let $\text{emb}(C) \stackrel{\text{def}}{=} (S, \mathbf{P})$ denote the DTMC such that for all $s, s' \in S$ we have $\mathbf{P}(s, s') \stackrel{\text{def}}{=} \frac{\mathbf{R}(s, s')}{\mathbf{u}(C)}$.

The following lemma states how values of CTMCs can be computed using the embedded discrete-time model.

Lemma 2: Let $C = (S, \mathbf{R})$ be a CTMC with a reward structure $\mathbf{r} = (\mathbf{r}_c, \mathbf{r}_f)$. Let $\mathbf{u} = \mathbf{u}(C)$. Then for $\mathbf{t} \geq 0$ and all $s_0 \in S$ it is

$$\mathbf{V}(C, s_0, \mathbf{r}, \mathbf{t}) = \mathbf{V}(\text{emb}(C), s_0, \mathbf{r}, \mathbf{t}, \mathbf{u})$$

Proof: By definition, for $s_0 \in S$ it is

$$\mathbf{V}(C, s_0, \mathbf{r}, \mathbf{t}) = \mathbf{E} \left[\underbrace{\int_0^{\mathbf{t}} \mathbf{r}_c(X_u^{C, s_0}) du}_{\text{accumulated}} \right] + \mathbf{E} \left[\underbrace{\mathbf{r}_f(X_{\mathbf{t}}^{C, s_0})}_{\text{final}} \right].$$

We have

$$\begin{aligned} & \text{final} \\ &= \mathbf{E} \left[\mathbf{r}_f(X_{\mathbf{t}}^{C, s_0}) \right] \\ &= \sum_{s \in S} \Pr(X_{\mathbf{t}}^{C, s_0} = s) \mathbf{r}_f(s) \\ &= \sum_{s \in S} \left(\sum_{i=0}^{\infty} \pi^{\text{emb}(C)}(s_0, i, s) \phi_{\mathbf{u}\mathbf{t}}(i) \right) \mathbf{r}_f(s) \\ &= \sum_{i=0}^{\infty} \phi_{\mathbf{u}\mathbf{t}}(i) \sum_{s \in S} \pi^{\text{emb}(C)}(s_0, i, s) \mathbf{r}_f(s). \end{aligned}$$

Further, Kwiatkowska et al. [53, Theorem 1] have shown that

$$\text{accumulated} = \sum_{i=0}^{\infty} \psi_{\mathbf{u}\mathbf{t}}(i) \sum_{s \in S} \pi^{\text{emb}(C)}(s_0, i, s) \frac{\mathbf{r}_c(s)}{\mathbf{u}}.$$

Thus,

$$\begin{aligned} \mathbf{V}(C, s_0, \mathbf{r}, \mathbf{t}) &= \sum_{i=0}^{\infty} \left(\phi_{\mathbf{u}\mathbf{t}}(i) \sum_{s \in S} \pi^{\text{emb}(C)}(s_0, i, s) \mathbf{r}_f(s) \right. \\ &\quad \left. + \psi_{\mathbf{u}\mathbf{t}}(i) \sum_{s \in S} \pi^{\text{emb}(C)}(s_0, i, s) \frac{\mathbf{r}_c(s)}{\mathbf{u}} \right) \\ &= \mathbf{V}(\text{emb}(C), s_0, \mathbf{r}, \mathbf{t}, \mathbf{u}). \end{aligned}$$

\square

We can now show that the restricted class CR suffices to obtain optimal values.

Lemma 3: Given a CTMDP $C = (S, \text{Act}, \mathbf{P})$ and $\sigma_{hr} \in \Sigma_{HR}$, there is $\sigma_{cr} \in \Sigma_{CR}$ such that $\mathbf{V}(X^{C, \sigma_{hr}, s_0}, \mathbf{r}, \mathbf{t}) = \mathbf{V}(X^{C, \sigma_{cr}, s_0}, \mathbf{r}, \mathbf{t})$. Further, for all $\sigma'_{cr} \in \Sigma_{CR}$ we can find $\sigma'_{hr} \in \Sigma_{HR}$ such that $\mathbf{V}(X^{C, \sigma'_{cr}, s_0}, \mathbf{r}, \mathbf{t}) = \mathbf{V}(X^{C, \sigma'_{hr}, s_0}, \mathbf{r}, \mathbf{t})$.

Proof: Consider a CTMDP $C = (S, \text{Act}, \mathbf{P})$ and $\sigma_{hr} \in \Sigma_{HR}$. By Lemma 1, we can find a scheduler $\sigma_{cr} \in \Sigma_{CR}$ such that

$$\pi^{\text{emb}(C), \sigma_{hr}} = \pi^{\text{emb}(C), \sigma_{cr}}. \quad (1)$$

Define

- $\mathbf{r}^{hr} \stackrel{\text{def}}{=} (\mathbf{r}_c^{hr}, \mathbf{r}_f^{hr})$ with
- $\mathbf{r}_c^{hr}(\beta, s) \stackrel{\text{def}}{=} \mathbf{r}_c(s)$ and
- $\mathbf{r}_f^{hr}(\beta, s) \stackrel{\text{def}}{=} \mathbf{r}_f(s)$

and let

- $\mathbf{r}^{cr} \stackrel{\text{def}}{=} (\mathbf{r}_c^{cr}, \mathbf{r}_f^{cr})$ with
- $\mathbf{r}_c^{cr}(s, n) \stackrel{\text{def}}{=} \mathbf{r}_c(s)$ and
- $\mathbf{r}_f^{cr}(s, n) \stackrel{\text{def}}{=} \mathbf{r}_f(s)$

for $\beta \in (S \times \text{Act})^*$, $n \in \mathbb{N}$ and $s \in S$. Then for all $s_0 \in S$ we have

$$\begin{aligned}
& \mathbf{E}[\mathbf{r}_f(X_t^{C, \sigma_{hr}, s_0})] \\
&= \mathbf{E}[\mathbf{r}_f(f(X_t^{C, \sigma_{hr}, s_0}))] \\
&= \sum_{s \in S} \Pr(f(X_t^{C, \sigma_{hr}, s_0}) = s) \mathbf{r}_f(s) \\
&= \sum_{s \in S} \Pr(\exists \beta \in (S \times \text{Act})^*. X_t^{C, \sigma_{hr}, s_0} = (\beta, s)) \mathbf{r}_f(s) \\
&= \sum_{\substack{s \in S \\ \beta \in (S \times \text{Act})^*}} \Pr(X_t^{C, \sigma_{hr}, s_0} = (\beta, s)) \mathbf{r}_f(s) \\
&= \sum_{\substack{s \in S \\ \beta \in (S \times \text{Act})^*}} \Pr(X_t^{C, \sigma_{hr}, s_0} = (\beta, s)) \mathbf{r}_f^{hr}(\beta, s) \\
&= \mathbf{E}[\mathbf{r}_f^{hr}(X_t^{C, \sigma_{hr}, s_0})]
\end{aligned}$$

and similarly

$$\begin{aligned}
& \mathbf{E} \left[\int_0^t \mathbf{r}_c(X_u^{C, \sigma_{hr}, s_0}) du \right] \\
&= \int_0^t \mathbf{E} [\mathbf{r}_c(X_u^{C, \sigma_{hr}, s_0})] du \\
&= \int_0^t \mathbf{E} [\mathbf{r}_c^{hr}(X_u^{C, \sigma_{hr}, s_0})] du \\
&= \mathbf{E} \left[\int_0^t \mathbf{r}_c^{hr}(X_u^{C, \sigma_{hr}, s_0}) du \right]
\end{aligned}$$

and in turn

$$\mathbf{V}(X^{C, \sigma_{hr}, s_0}, \mathbf{r}, \mathbf{t}) = \mathbf{V}(X^{C, \sigma_{hr}, s_0}, \mathbf{r}^{hr}, \mathbf{t}). \quad (2)$$

In the same way, using $n \in \mathbb{N}$ instead of $\beta \in (S \times \text{Act})^*$, one can show

$$\mathbf{V}(X^{C, \sigma_{cr}, s_0}, \mathbf{r}, \mathbf{t}) = \mathbf{V}(X^{C, \sigma_{cr}, s_0}, \mathbf{r}^{cr}, \mathbf{t}). \quad (3)$$

Notice that for $\sigma \in \Sigma_{CR} \cup \Sigma_{HR}$ it is

$$\text{emb}(C_\sigma) = \text{emb}(C)_{\sigma}. \quad (4)$$

In addition,

$$\begin{aligned}
& \mathbf{V}(\text{emb}(C)_{\sigma_{hr}}, s_0, \mathbf{r}^{hr}, \mathbf{t}, \mathbf{u}) \\
&= \sum_{i=0}^{\infty} \left(\phi_{\mathbf{ut}}(i) \sum_{s \in (S \times \text{Act})^* \times S} \pi^{\text{emb}(C)_{\sigma_{hr}}}(s_0, i, s) \cdot \mathbf{r}_f^{hr}(s) \right. \\
&\quad \left. + \psi_{\mathbf{ut}}(i) \sum_{s \in (S \times \text{Act})^* \times S} \pi^{\text{emb}(C)_{\sigma_{hr}}}(s_0, i, s) \frac{\mathbf{r}_c^{hr}(s)}{\mathbf{u}} \right) \\
&= \sum_{i=0}^{\infty} \left(\phi_{\mathbf{ut}}(i) \sum_{s \in S} \pi^{\text{emb}(C)_{\sigma_{hr}}}(s_0, i, s) \cdot \mathbf{r}_f(s) \right. \\
&\quad \left. + \psi_{\mathbf{ut}}(i) \sum_{s \in S} \pi^{\text{emb}(C)_{\sigma_{hr}}}(s_0, i, s) \frac{\mathbf{r}_c(s)}{\mathbf{u}} \right) \\
&\stackrel{\text{Eqn. 1}}{=} \sum_{i=0}^{\infty} \left(\phi_{\mathbf{ut}}(i) \sum_{s \in S} \pi^{\text{emb}(C)_{\sigma_{cr}}}(s_0, i, s) \cdot \mathbf{r}_f(s) \right. \\
&\quad \left. + \psi_{\mathbf{ut}}(i) \sum_{s \in S} \pi^{\text{emb}(C)_{\sigma_{cr}}}(s_0, i, s) \frac{\mathbf{r}_c(s)}{\mathbf{u}} \right)
\end{aligned} \quad (5)$$

$$\begin{aligned}
&= \sum_{i=0}^{\infty} \left(\phi_{\mathbf{ut}}(i) \sum_{s \in S \times \mathbb{N}} \pi^{\text{emb}(C)_{\sigma_{cr}}}((s_0, 0), i, s) \cdot \mathbf{r}_f^{cr}(s) \right. \\
&\quad \left. + \psi_{\mathbf{ut}}(i) \sum_{s \in S \times \mathbb{N}} \pi^{\text{emb}(C)_{\sigma_{cr}}}((s_0, 0), i, s) \frac{\mathbf{r}_c^{cr}(s)}{\mathbf{u}} \right) \\
&= \mathbf{V}(\text{emb}(C)_{\sigma_{cr}}, s_0, \mathbf{r}^{cr}, \mathbf{t}, \mathbf{u}).
\end{aligned}$$

From these facts, we have

$$\begin{aligned}
& \mathbf{V}(X^{C, \sigma_{hr}, s_0}, \mathbf{r}, \mathbf{t}) \\
&\stackrel{\text{Eqn. 2}}{=} \mathbf{V}(X^{C, \sigma_{hr}, s_0}, \mathbf{r}^{hr}, \mathbf{t}) \\
&\stackrel{\text{Lemma 2}}{=} \mathbf{V}(\text{emb}(C)_{\sigma_{hr}}, s_0, \mathbf{r}^{hr}, \mathbf{t}, \mathbf{u}) \\
&\stackrel{\text{Eqn. 4}}{=} \mathbf{V}(\text{emb}(C)_{\sigma_{hr}}, s_0, \mathbf{r}^{hr}, \mathbf{t}, \mathbf{u}) \\
&\stackrel{\text{Eqn. 5}}{=} \mathbf{V}(\text{emb}(C)_{\sigma_{cr}}, s_0, \mathbf{r}^{cr}, \mathbf{t}, \mathbf{u}) \\
&\stackrel{\text{Eqn. 4}}{=} \mathbf{V}(\text{emb}(C)_{\sigma_{cr}}, s_0, \mathbf{r}^{cr}, \mathbf{t}, \mathbf{u}) \\
&\stackrel{\text{Lemma 2}}{=} \mathbf{V}(X^{C, \sigma_{cr}, s_0}, \mathbf{r}^{cr}, \mathbf{t}) \\
&\stackrel{\text{Eqn. 3}}{=} \mathbf{V}(X^{C, \sigma_{cr}, s_0}, \mathbf{r}, \mathbf{t}).
\end{aligned} \quad (6)$$

If we start with a CR σ'_{cr} , we can define the HR σ'_{hr} such that for $\beta \in (S \times \text{Act})^n$ with $n \in \mathbb{N}$ we have $\sigma'_{hr}(\beta, s) \stackrel{\text{def}}{=} \sigma'_{cr}(s, n)$ and then the result can be shown in the same way. \square

From Lemma 3, we can conclude that the maximum is obtained by a scheduler in Σ_{CR} .

Corollary 2: Given a CTMDP C and reward structure \mathbf{r} , for all $s_0 \in S$ it is

$$\mathbf{V}^{\max}(C, s_0, \mathbf{r}, \mathbf{t}) = \max_{\sigma \in \Sigma_{CR}} \mathbf{V}(X^{C, \sigma, s_0}, \mathbf{r}, \mathbf{t}).$$

Because of Corollary 2, we only have to show that Algorithm 1 computes the maximum over all $\sigma \in \Sigma_{CR}$ up to the specified precision.

We first show that to compute the value of a given CTMDP for a given scheduler up to a required precision, it suffices to consider a limited number of steps in the embedded model.

Lemma 4: Given a CTMDP $C = (S, \text{Act}, \mathbf{P})$ with a reward structure $\mathbf{r} = (\mathbf{r}_c, \mathbf{r}_f)$, a precision $\varepsilon > 0$ and k such that

$$\sum_{n=0}^k \psi_{\mathbf{ut}}(n) > \mathbf{ut} - \frac{\varepsilon \mathbf{u}}{2 \mathbf{r}_c^{\max}} \text{ and } \psi_{\mathbf{ut}}(k) \cdot \mathbf{r}_f^{\max} < \frac{\varepsilon}{2},$$

then for all schedulers $\sigma \in \Sigma_{CR}$ and all $s_0 \in S$ we have

$$\begin{aligned}
& \sum_{i=k+1}^{\infty} \left(\phi_{\mathbf{ut}}(i) \sum_{s \in S} \pi^{\text{emb}(C)_{\sigma}}(s_0, i, s) \mathbf{r}_f(s) \right. \\
&\quad \left. + \psi_{\mathbf{ut}}(i) \sum_{s \in S} \pi^{\text{emb}(C)_{\sigma}}(s_0, i, s) \frac{\mathbf{r}_c(s)}{\mathbf{u}} \right) < \varepsilon.
\end{aligned}$$

Proof: Consider a CTMC $C' = (S', \mathbf{P}')$ with reward structure $\mathbf{r}' = (\mathbf{r}'_c, \mathbf{r}'_f)$ and

$$\mathbf{V}(C', s_0, \mathbf{r}', \mathbf{t}) = \mathbf{E} \left[\underbrace{\int_0^{\mathbf{t}} \mathbf{r}'_c(X_u^{C', s_0}) du}_{\text{accumulated}} \right] + \underbrace{\mathbf{E} [\mathbf{r}'_f(X_{\mathbf{t}}^{C', s_0})]}_{\text{final}}$$

for any $s_0 \in S$. It is known [53, remark below Theorem 2] that if

$$\sum_{n=0}^k \psi_{\mathbf{ut}}(n) > \mathbf{ut} - \frac{\varepsilon \mathbf{u}}{2\mathbf{r}'_c^{\max}}$$

then

$$\begin{aligned} & \text{accumulated} \\ & \stackrel{\text{def}}{=} \sum_{n=k+1}^{\infty} \psi_{\mathbf{ut}}(i) \sum_{s \in S'} \pi^{\text{emb}(C')}(s_0, i, s) \frac{\mathbf{r}'_c(s)}{\mathbf{u}} \\ & < \frac{\varepsilon}{2}, \end{aligned}$$

and if

$$\psi_{\mathbf{ut}}(k) \cdot \mathbf{r}'_f^{\max} < \frac{\varepsilon}{2}$$

then

$$\begin{aligned} & \text{final} \\ & \stackrel{\text{def}}{=} \sum_{i=k+1}^{\infty} \phi_{\mathbf{ut}}(i) \sum_{s \in S'} \pi^{\text{emb}(C')}(s_0, i, s) \cdot \mathbf{r}'_f(s) \\ & \leq \sum_{i=k+1}^{\infty} \phi_{\mathbf{ut}}(i) \sum_{s \in S'} \pi^{\text{emb}(C')}(s_0, i, s) \cdot \mathbf{r}'_f^{\max} \\ & = \sum_{i=k+1}^{\infty} \phi_{\mathbf{ut}}(i) \mathbf{r}'_f^{\max} \\ & = \psi_{\mathbf{ut}}(k) \cdot \mathbf{r}'_f^{\max} \\ & < \frac{\varepsilon}{2}. \end{aligned}$$

Thus,

$$\begin{aligned} & \sum_{i=k+1}^{\infty} \left(\phi_{\mathbf{ut}}(i) \sum_{s \in S} \pi^{\text{emb}(C')}(s_0, i, s) \mathbf{r}'_f(s) \right. \\ & \quad \left. + \psi_{\mathbf{ut}}(i) \sum_{s \in S} \pi^{\text{emb}(C')}(s_0, i, s) \frac{\mathbf{r}'_c(s)}{\mathbf{u}} \right) \\ & = \text{accumulated} + \text{final} \\ & < \varepsilon. \end{aligned} \tag{7}$$

Now with $\mathbf{r}' \stackrel{\text{def}}{=} (\mathbf{r}'_c, \mathbf{r}'_f)$ where $\mathbf{r}'_c(s, n) \stackrel{\text{def}}{=} \mathbf{r}_c(s)$ and $\mathbf{r}'_f(s, n) \stackrel{\text{def}}{=} \mathbf{r}_f(s)$, because $\mathbf{r}_c^{\max} = \mathbf{r}'_c^{\max}$ and $\mathbf{r}_f^{\max} = \mathbf{r}'_f^{\max}$ it is

$$\begin{aligned} & \sum_{i=k+1}^{\infty} \left(\phi_{\mathbf{ut}}(i) \sum_{s \in S} \pi^{\text{emb}(C), \sigma}(s_0, i, s) \cdot \mathbf{r}_f(s) \right. \\ & \quad \left. + \psi_{\mathbf{ut}}(i) \sum_{s \in S} \pi^{\text{emb}(C), \sigma}(s_0, i, s) \frac{\mathbf{r}_c(s)}{\mathbf{u}} \right) \\ & = \sum_{i=k+1}^{\infty} \left(\phi_{\mathbf{ut}}(i) \sum_{s \in S \times \mathbb{N}} \pi^{\text{emb}(C), \sigma}(s_0, i, s) \cdot \mathbf{r}'_f(s) \right. \\ & \quad \left. + \psi_{\mathbf{ut}}(i) \sum_{s \in S \times \mathbb{N}} \pi^{\text{emb}(C), \sigma}(s_0, i, s) \frac{\mathbf{r}'_c(s)}{\mathbf{u}} \right) \\ & < \varepsilon. \end{aligned}$$

Algorithm 4: Compute value of $C = (S, \text{Act}, \mathbf{R})$, reward structure $(\mathbf{r}_c, \mathbf{r}_f)$ and CR σ up to precision ε .

```

1 let  $k$  s.t.  $\sum_{n=0}^k \psi_{\mathbf{ut}}(n) > \mathbf{ut} - \frac{\varepsilon \mathbf{u}}{2\mathbf{r}_c^{\max}} \wedge \psi_{\mathbf{ut}}(k) \cdot \mathbf{r}_f^{\max} < \frac{\varepsilon}{2}$ 
2  $C' = (S, \text{Act}, \mathbf{P}) := \text{emb}(C)$ 
3 forall  $s \in S$  do  $q_{k+1}(s) := 0$ 
4 forall  $i = k, k-1, \dots, 0$  do
5   forall  $s \in S$  do
6      $m := \sum_{\alpha \in \text{Act}} \sigma(s, k)(\alpha) \sum_{s' \in S} \mathbf{P}(s, \alpha, s') q_{i+1}(s')$ 
7      $q_i(s) := m + \phi_{\mathbf{ut}}(i) \cdot \mathbf{r}_f(s) + \psi_{\mathbf{ut}}(i) \cdot \frac{\mathbf{r}_c(s)}{\mathbf{u}}$ 
8 return  $q_0$ 

```

We can show that Algorithm 4 computes the values of a CTMDP given a certain scheduler up to a required precision.

Lemma 5: Consider a CTMDP $C = (S, \text{Act}, \mathbf{R})$, a time bound \mathbf{t} , a scheduler $\sigma \in \Sigma_{CR}$, and let q be the return value of Algorithm 4. Then for all $s_0 \in S$ it is $|q(s_0) - \mathbf{V}(X^{C, \sigma, s_0}, \mathbf{r}, \mathbf{t})| < \varepsilon$.

Proof: Consider the values $q = q_0$ returned by Algorithm 4. It is

$$\begin{aligned} & q(s_0) \\ & = \phi_{\mathbf{ut}}(0) \cdot \mathbf{r}_f(s_0) + \psi_{\mathbf{ut}}(0) \frac{\mathbf{r}_c(s_0)}{\mathbf{u}} \\ & \quad + \sum_{\alpha_0 \in \text{Act}} \sigma(s_0, 0)(\alpha_0) \sum_{s_1 \in S} \mathbf{P}(s_0, \alpha_0, s_1) \\ & \quad \cdot (\phi_{\mathbf{ut}}(1) \cdot \mathbf{r}_f(s_1) + \psi_{\mathbf{ut}}(1) \frac{\mathbf{r}_c(s_1)}{\mathbf{u}} \\ & \quad + \sum_{\alpha_1 \in \text{Act}} \sigma(s_1, 1)(\alpha_1) \sum_{s_2 \in S} \mathbf{P}(s_1, \alpha_1, s_2) \dots \\ & = \phi_{\mathbf{ut}}(0) \cdot \mathbf{r}_f(s_0) + \psi_{\mathbf{ut}}(0) \frac{\mathbf{r}_c(s_0)}{\mathbf{u}} \\ & \quad + \sum_{\alpha_0 \in \text{Act}} \sigma(s_0, 0)(\alpha_0) \sum_{s_1 \in S} \mathbf{P}(s_0, \alpha_0, s_1) (\phi_{\mathbf{ut}}(1) \cdot \mathbf{r}_f(s_0) \\ & \quad + \psi_{\mathbf{ut}}(1) \frac{\mathbf{r}_c(s_0)}{\mathbf{u}}) \\ & \quad + \sum_{\alpha_0 \in \text{Act}} \sigma(s_0, 0)(\alpha_0) \sum_{s_1 \in S} \mathbf{P}(s_0, \alpha_0, s_1) \sum_{\alpha_1 \in \text{Act}} \sigma(s_1, 1) \dots \tag{8} \\ & \quad + \dots \\ & \stackrel{\text{Cor. 1}}{=} \sum_{i=0}^k \left(\phi_{\mathbf{ut}}(i) \sum_{s \in S} \pi^{\text{emb}(C), \sigma}(s_0, i, s) \mathbf{r}_f(s) \right. \\ & \quad \left. + \psi_{\mathbf{ut}}(i) \sum_{s \in S} \pi^{\text{emb}(C), \sigma}(s_0, i, s) \frac{\mathbf{r}_c(s)}{\mathbf{u}} \right) \\ & = \mathbf{V}(\text{emb}(C), s_0, \mathbf{r}, \mathbf{t}, \mathbf{u}) - \\ & \quad \sum_{i=k+1}^{\infty} \left(\phi_{\mathbf{ut}}(i) \sum_{s \in S} \pi^{\text{emb}(C), \sigma}(s_0, i, s) \mathbf{r}_f(s) \right. \\ & \quad \left. + \psi_{\mathbf{ut}}(i) \sum_{s \in S} \pi^{\text{emb}(C), \sigma}(s_0, i, s) \frac{\mathbf{r}_c(s)}{\mathbf{u}} \right) \\ & \stackrel{\text{Lemma 4}}{=} \mathbf{V}(\text{emb}(C), s_0, \mathbf{r}, \mathbf{t}, \mathbf{u}) - \varepsilon' \end{aligned}$$

for some ε' with $0 \leq \varepsilon' < \varepsilon$, and thus $|q(s_0) - \mathbf{V}(X^{C, \sigma, s_0}, \mathbf{r}, \mathbf{t})| < \varepsilon$. \square

The value obtained from Algorithm 1 will be no smaller than the one obtained from applying Algorithm 4 on an arbitrary scheduler.

Lemma 6: Consider a CTMDP $C = (S, Act, \mathbf{R})$, a time bound \mathbf{t} , an arbitrary scheduler $\sigma \in \Sigma_{CR}$, let q be the return value of Algorithm 4 and let q' be the return value of Algorithm 1. Then for all $s_0 \in S$ it is $q(s_0) \leq q'(s_0)$.

Proof: Let q_i be as given in Algorithm 4 and let q'_i be the corresponding vector of Algorithm 1. We show the lemma by backward induction on the program variable i .

Induction start: $i = k + 1$: Before the main loop at the lines 3, both algorithms assign $q_{k+1}(s) = q'_{k+1}(s) = 0$ for all $s \in S$.

Induction assumption: Assume it is $q_{i+1}(s) \leq q'_{i+1}(s)$ at the beginning of the main loops, that is before lines 4.

Induction step: Consider m of Algorithm 4 and corresponding m' of Algorithm 1 after the assignment to this variable at line 6. It is

$$\begin{aligned} m &= \sum_{\alpha \in Act(s)} \sigma(s, k)(\alpha) \sum_{s' \in S} \mathbf{P}(s, \alpha, s') q_{i+1}(s') \\ &\leq \max_{\alpha \in Act(s)} \sum_{s' \in S} \mathbf{P}(s, \alpha, s') q_{i+1}(s') \\ &\stackrel{\text{Ass.}}{\leq} \max_{\alpha \in Act(s)} \sum_{s' \in S} \mathbf{P}(s, \alpha, s') q'_{i+1}(s') \\ &= m'. \end{aligned}$$

Because lines 7 are identical in both algorithms, also $q_i \leq q'_i$ at the end of the main loops. \square

With these preparations, we can now prove the first part of Proposition 1.

Lemma 7: Consider a CTMDP $C = (S, Act, \mathbf{R})$, a reward structure \mathbf{r} , a time bound \mathbf{t} and let q' be the return value of Algorithm 1. Then for all $s_0 \in S$ it is $|q'(s_0) - \mathbf{V}^{\max}(C, s_0, \mathbf{r}, \mathbf{t})| < \varepsilon$.

Proof: Let $\sigma \in \Sigma_{CR}$ be such that

$$\mathbf{V}(X^{C, \sigma, s_0}, \mathbf{r}, \mathbf{t}) = \mathbf{V}^{\max}(C, s_0, \mathbf{r}, \mathbf{t}). \quad (9)$$

Then because of Lemma 5 it is $|q(s_0) - \mathbf{V}(X^{C, \sigma, s_0}, \mathbf{r}, \mathbf{t})| < \varepsilon$ where q is the return value of Algorithm 4. Because of Lemma 6 we know that $q(s_0) \leq q'(s_0)$. By adding the assignment

$$\sigma'_{cd}(s, i) := \arg \max_{\alpha \in Act(s)} \sum_{s' \in S} \mathbf{P}(s, \alpha, s') q_{i+1}(s')$$

into the inner loop of Algorithm 1 after Line 7, we can obtain a prefix of the scheduler $\sigma'_{cd} \in \Sigma_{CD}$. Consider $\sigma'_{cr} \in \Sigma_{CR}$ such that $\sigma'_{cr}(s, i)(\alpha) = 1$ if $\sigma'_{cd}(s, i) = \alpha$ and $\sigma'_{cr}(s, i)(\alpha) = 0$ else. It can easily be shown that applying Algorithm 4 on σ'_{cr} also yields the value q' . Thus again by Lemma 5 we have $|q'(s_0) - \mathbf{V}(X^{C, \sigma', s_0}, \mathbf{r}, \mathbf{t})| < \varepsilon$. \square

We can now show that deterministic schedulers suffice, by using the fact that Algorithm 1 indeed maximises only over this class.

Lemma 8: Let $C = (S, Act, \mathbf{R})$ be a CTMDP with reward structure $\mathbf{r} = (\mathbf{r}_c, \mathbf{r}_f)$. Then there exists $\sigma \in \Sigma_{CD}$ such that $\mathbf{V}^{\max}(C, s_0, \mathbf{r}, \mathbf{t}) = \mathbf{V}(X^{C, \sigma, s_0}, \mathbf{r}, \mathbf{t})$

Proof: Assume the lemma does not hold. Then for each $\sigma \in \Sigma_{CD}$ there is a $\varepsilon > 0$ such that for some state $s_0 \in S$ we have $|\mathbf{V}(X^{C, \sigma, s_0}, \mathbf{r}, \mathbf{t}) - \mathbf{V}^{\max}(C, s_0, \mathbf{r}, \mathbf{t})| \geq \varepsilon$. Now consider $\sigma'_{cd} \in \Sigma_{CD}$ obtained by Algorithm 1 as in the proof of Lemma 7

using the same required precision ε . By the correctness of Algorithm 1, we then have $|\mathbf{V}(X^{C, \sigma', s_0}, \mathbf{r}, \mathbf{t}) - \mathbf{V}^{\max}(C, s_0, \mathbf{r}, \mathbf{t})| < \varepsilon$. This contradicts the assumption, because with the extension in the proof of Lemma 7, the algorithm also computes a CD which obtains this precision. \square

The idea of using the fact that the optimising algorithm computes a scheduler of a more restricted class than the class considered was adapted from various proofs for similar problems in the discrete-time setting [40].

APPENDIX B

PROOF OF PROPOSITION 2

Consider $(S, \mathbf{P}) = \mathcal{D} \stackrel{\text{def}}{=} \text{emb}(C)$ and $(S, \mathbf{P}') = \mathcal{D}' \stackrel{\text{def}}{=} \text{emb}(C')$. We define the HR $\sigma: ((\mathbb{A} \times Act)^* \times \mathbb{A}) \rightarrow \text{Distr}(Act)$ so that for $\beta = \beta_0 \alpha_{s_0} \beta_1 \alpha_{s_1} \dots \alpha_{s_{n-1}} \beta_n$ we have

$$\sigma(\beta)(\alpha_s) \stackrel{\text{def}}{=} \Pr(X_n^{\mathcal{D}, s_0} = s \mid X_n^{\mathcal{D}, s_0} \wedge \bigwedge_{i=0}^{n-1} X_i^{\mathcal{D}, s_0} = s_i).$$

By induction, for all $n \in \mathbb{N}$ and $\beta \in \mathbb{A}$ we have

$$\Pr(X_n^{\mathcal{D}, s_0} \in \beta) = \Pr(X_n^{\mathcal{D}', \sigma, \beta_0} \in \beta).$$

Thus, using Definition 22, Definition 24, and the definition of the reward structures \mathbf{r}, \mathbf{r}' it is

$$\mathbf{V}(\mathcal{D}, s_0, \mathbf{r}, \mathbf{t}, \mathbf{u}) \leq \mathbf{V}(\mathcal{D}', \beta_0, \mathbf{r}, \mathbf{t}, \mathbf{u}),$$

and thus using Lemma 2 it is

$$\mathbf{V}(X^{C, s_0}, \mathbf{r}, \mathbf{t}) \leq \mathbf{V}(X^{C', \sigma, \beta_0}, \mathbf{r}', \mathbf{t}) \leq \mathbf{V}^{\max}(C', \beta_0, \mathbf{r}', \mathbf{t}).$$

APPENDIX C

PROOF OF PROPOSITION 3

We only show that by using a finer abstraction the maximal bound cannot increase. The case for the minimal bound is likewise. We will show that for each $\varepsilon > 0$ it is

$$\mathbf{V}^{\max}(C, \beta_t, \mathbf{r}, \mathbf{t}) + \varepsilon \geq \mathbf{V}^{\max}(C', \beta'_{t,j}, \mathbf{r}', \mathbf{t}). \quad (10)$$

This then also shows that the inequation holds for $\varepsilon = 0$: If $\mathbf{V}^{\max}(C, \beta_t, \mathbf{r}, \mathbf{t}) < \mathbf{V}^{\max}(C', \beta'_{t,j}, \mathbf{r}', \mathbf{t})$ then $\varepsilon' \stackrel{\text{def}}{=} \mathbf{V}^{\max}(C', \beta'_{t,j}, \mathbf{r}', \mathbf{t}) - \mathbf{V}^{\max}(C, \beta_t, \mathbf{r}, \mathbf{t})$ is positive. By subtracting $\mathbf{V}^{\max}(C, \beta_t, \mathbf{r}, \mathbf{t})$ from Equation 10 we have

$$\varepsilon \geq \varepsilon'.$$

As this equation must hold for all ε , for instance $\frac{\varepsilon'}{2}$, this is a contradiction.

We show Equation 10. To do so, we show by a backward induction on Algorithm 1 that for the value q_0 obtained for C and the value q' obtained for C' we have $q \geq q'_0$, using a precision of ε . By the precision guarantee of the algorithm, this in turn shows the equation.

Induction start: $i = k + 1$: Before the main loop, at the Line 3 both runs assign $q_{k+1}(\beta_t) = q'_{k+1}(\beta'_{t,j}) = 0$ for all $\beta_t \in \mathbb{P}, \beta'_{t,j} \in \mathbb{P}'$.

Induction assumption: Assume it is $q_{i+1}(\beta_t) \geq q'_{i+1}(\beta'_{t,j})$ for all $\beta_t \in \mathbb{P}, \beta'_{t,j} \in \mathbb{P}'$ with $\beta_t = \bigcup_{j=1}^u \beta'_{t,j}$ at the beginning of the main loops, that is before Line 4.

Induction step: Consider m for β_t of C and corresponding m' for $\beta'_{t,j}$ of C' after the assignment to this variable at Line 6. Let

$(\hat{\alpha}', \alpha')$ be a maximising decision for some $\mathfrak{z}'_{t,j}$ at this line. By the definition of the abstraction, for \mathfrak{z}_t we find a corresponding $(\hat{\alpha}, \alpha)$ such that

- $Dom(\hat{\alpha}') = Dom(\hat{\alpha})$,
- for each $c \in Dom(\hat{\alpha}')$ we have $\hat{\alpha}'(c) = (\mathfrak{z}'_{v,l}, \mathbf{I})$ and $\hat{\alpha}(c) = (\mathfrak{z}_v, \mathbf{I})$ with $\mathfrak{z}'_{v,l} \subseteq \mathfrak{z}_v$.

Then it is

$$\begin{aligned}
m' &= \max_{(\hat{\alpha}', \alpha') \in Act(\mathfrak{z}'_{t,j})} \sum_{\mathfrak{z}'_{v,l} \in \mathfrak{P}'} \mathbf{P}(\mathfrak{z}_{t,j}, (\hat{\alpha}', \alpha'), \mathfrak{z}'_{v,l}) q_{i+1}(\mathfrak{z}'_{v,l}) \\
&= \sum_{\mathfrak{z}'_{v,l} \in \mathfrak{P}'} \mathbf{P}(\mathfrak{z}'_{t,j}, (\hat{\alpha}', \alpha'), \mathfrak{z}'_{v,l}) q_{i+1}(\mathfrak{z}'_{v,l}) \\
&\stackrel{Ass.}{\leq} \sum_{\mathfrak{z}_v \in \mathfrak{P}} \mathbf{P}(\mathfrak{z}_t, (\hat{\alpha}, \alpha), \mathfrak{z}_v) q_{i+1}(\mathfrak{z}_v) \\
&\leq \max_{(\hat{\alpha}, \alpha) \in Act(\mathfrak{z}_t)} \sum_{\mathfrak{z}_v \in \mathfrak{P}} \mathbf{P}(\mathfrak{z}_t, (\hat{\alpha}, \alpha), \mathfrak{z}_v) q_{i+1}(\mathfrak{z}_v) \\
&= m.
\end{aligned}$$

The rewards for the refined abstraction cannot be larger than the ones for the coarser one. Thus, after Line 7, we still have $q_i \geq q'_i$ at the end each iteration. ■